НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою з програми “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”
Баланюка Северина

на тему: Система моделювання квантових обчислень

Виконав : студент __4__ курсу, групи ІО-з81
(шифр групи)

Северина Баланюка Олександровича
(прізвище, ім’я, по батькові) (підпис)

Керівник професор, д.т.н.,Луцький Г.М.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) професор, д.т.н., Сімоненко В. П.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Баланюка Северина Олександровича

1. Тема проєкту *Система моделювання квантових обчислень*
керівник проєкту професор_д.т.н.,Луцький Г.М
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 11 травня 2022 року №1139-с

2. Термін здачі студентом закінченого проєкту *13 Червня 2022 р.*.

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд теоритичних данних та принципів створення квантових обчислювальних систем.

Розділ 2. Огляд дизайну та порівняльний аналіз симуляторів

Розділ 3. Деталі розробки системи.

4. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

5. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль			

7. Дата видачі завдання «30» серпня 2021 р.

Календарний план

п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
	Затвердження теми проєкту	*10.12.2021-15.12.2022*	
	Вивчення та аналіз завдання	*15.12.2021-15.03.2022*	
.	*Розробка архітектури та загальної структури системи*	*15.03.2021-25.03.2022*	
.	*Розробка структур окремих підсистем*	*25.03.2021-5.04.2022*	
	Програмна реалізація	*5.04.2021-15.04.2022*	
	Оформлення пояснювальної	*15.04.2021-20.05.2022*	
	Захист програмного продукту	*25.04.2022*	
	Передзахист	*23.05.2022*	
	Захист	*20.06.2022*	

Студент-дипломник _____ Северин БАЛАНЮК
(підпис)

Керівник проєкту _____ Георгій ЛУЦЬКИЙ.
(підпис)

АНОТАЦІЯ

У цьому проекті було детально розглянуто системи моделювання квантвих обчислень. Продемосровано актуальність доследження та її проблематика, також зроблено ознайомлення з усіма основними технологіями і способами для моделювання квантових обчислень на звичнайних електронно обчислювальних машинах. Зроблено порівняння двох відомих та найактулальніших способи моделювання таких обчисленнь. В результаті написана программа яка демонструє принцип та ефективніть данних способів моделювання та поянені головні принципи на яких будується така система та її симуляція. Димпломний проект ставить перед собою задачу дослідити новий підход до вирішення існуючих математичних проблем та розглядає перспективи використання квантових компьютерів для вірішення більш кладних та массивних задач в різних сферах людської діяльнсоті.

Ключові слова: симуляція квантових обчислень, квантовий компьютер, Qiskit, Python.

ANNOTATION

In this project, quantum computation modeling systems were considered in details. The relevance of the research and its problems are demonstrated, as well as the acquaintance with all the main technologies and methods for modeling quantum computing on conventional electronic computers. A comparison of two known and most reliable methods of modeling such calculations is made. As a result, a program was written that demonstrates the principle and effective data of modeling methods and explained the main principles on which such a system is built. The project aims to explore a new approach to solve mathematical problems and the prospects of using quantum computers to perform more complex and mass tasks in various fields of human activity.

Keywords: quantum computing simulation, quantum computer, Qiskit, Python.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Система моделювання	3		
			Квантових обчислень			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Система моделювання	76		
			Квантових обчислень			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Система моделювання	1		
			Квантових обчислень			
			Структурна схема			
			системи			
	А4	***ІАЛЦ.4672008.005 Д2***	Система моделювання	1		
			Квантових обчислень			
			Функціональна схема			
	А4	***ІАЛЦ.467200.006 Д3***	Система моделювання	1		
			Квантових обчислень			
			Алгоритм дій програмного			
			забезпечення			
	А4	***ІАЛЦ.467200.007 Д4***	Система моделювання	15		
			Квантових обчислень			
			Текст програмного коду			

					# *ІАЛЦ.467200.001 ОА*		
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*			
Розроб		Баланюк С.О			*Система моделювання квантових обчислень* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Луцький Г.М.					
						"КПІ" імені Сікорського ФІОТ ІО-з81	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система моделювання квантових обчислень*»

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на область дослідження та розробки систем для симуляції квантових обчислень на звичайниї ЕОМ та вирішення математичних проблем.

Областю застосування цієї системи є створення ефективних алгоритмів для вирішення математичних та фізичних проблем та симуляції процесів з великою кількістю перемінних.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «Комп'ютерні системи та мережі», який був затверджений факультетом "Інформатики та обчислювальної техніки" кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є дослідження та випробування кращої системи симуляції квантових обчислень та тестування її ефективності за допомогою алгоритмів та розробки нових способів обчислення задач.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий і інтуїтивно-зрозумілий алгоритм роботи системи.
- Надати можливість користувачам власноруч вибирати бажаний алгоритм для обчислення.
- Надати вичерпну та зрозумілу документацію для розробленого продукту.
- Надати інформацію щодо найкращої розробки по темі та майбутніх песпектив області.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Python 3.

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i5-8250U.
- ROM не менше ніж 32 ГБ.
- RAM не менше ніж 8 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2021-15.12.2021
Вивчення та аналіз завдання	15.12.2021-15.03.2022
Розробка архітектури та загальної структури системи	15.03.2022-25.03.2022
Розробка структур окремих частин системи	25.03.2022-5.04.2022
Програмна реалізація системи	5.04.2022-15.04.2022
Виправлення помилок	15.04.2022-20.05.2022
Оформлення пояснювальної записки	25.04.2022

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система моделювання квантових обчислень*»

Київ – 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

QC	(Quantum computer/quantum computing) квантовий компьтер
QKD	(quantum key distribution) – квантове розподілення ключів
QRAM	(Quantum random access memory) квантова пам'ять швидкого доступу
qubit	(quantum bit) кватовий біт
R&D	(research and development) дослідження та розробка
1Qgate.	Short for *one-qubit gate* (*one-qubit quantum logic gate*.) скорочення для одного кубітного входу
QD	(Quantum Dot) квантова точка
QFT	(Quantum Field Theory) теорія квантового поля
ala	(Algebraic approach) Алгебраїчний підхід
ana	(Analytical approach) Аналітичний підхід
CIM	(Classical mechanics) Класична механіка
QZE	(Quantum Zeno effect) Квантовий ефект Зенона
PBS	(Polarizing beam splitter) роздільник поляризованого світла
DEq	(Differential equation) диференційне рівняння
RAM	(random access memory) оперативна пам'ять

ВСТУП

Поштовх розвитку науки і техніки, що призвів до розвитку цивілізації, відкрив нові шляхи використання різних фізичних ресурсів для покращення життя людей, таких як матеріали, сили та енергія. Історія розвитку комп'ютерів є кульмінацією років технологічного прогресу, починаючи з ранніх ідей Чарза Беббіджа і в кінцевому підсумку створення першого комп'ютера німемцем Конардом Цейсе в 1941 році. Увесь шлях включення послідовності змін від одного типу фізичної реалізації до іншого від шертірні до реле, до різних клапанів, до транзисторів, до інтегральних схем, до мікросхем тощо. Дивно, але ефективний сучасний комп'ютер принципово нічим не відрізняється від своїх величезних 30-тонних предків, які мали приблизно 18 000 вакуумних лампам та 500 мільйонів різних провідників. Не дивлячись на те, що комп'ютери стали більш малі і значно швидше виконують свої задачі, залишається незмінним маніпулювання та інтерпретувати кодування двох бітів у кінечний результат розрахунків[1].

Наступним логічним кроком для розвитку науки і техніки я бачу формулювання ідеї використання квантовомеханічних систем для зберігання та обробки величзних масивів інформіції інформації, бо область квантових обчислень швидко розвивається, особливо після розробки кількох ефективних квантових алгоритмів, що вирішують важкорозв'язні класичні проблеми. Наприклад, відкриття алгоритму розкладання простих чисел Шорса [3] окреслило величезний потенціал квантових обчислень, факторинг простих чисел є основою кількох криптосистем, таких як широко використовувана схема безпеки Рівеста-Шаміра-Адлемана (RSA) [4]. Пізніше алгоритм пошуку Гроверса [5] продемонстрував квадратичне прискорення порівняно з відповідною класичною реалізацією. Відкриття цих алгоритмів прискорило розробку різних реалізацій фізичних квантових комп'ютерів з використанням різних технологій, таких як квантова

електродинаміка порожнини (QED) [6], захоплені іони [7] або твердотільні системи [8].

Незважаючи на швидке покращення часу когерентності кубітів і точності виконання отворів у різних технологіях, поточні експериментальні платформи надають обмежену кількість кубітів і обмежені оперативні можливості. Визначаючи та будуючи архітектуру для квантового комп'ютера, необхідно розуміти, як адресувати та контролювати більшу кількість кубітів. На найвищому рівні розробники алгоритмів формулюють квантові алгоритми, такі як алгоритм факторингу Шора, мовою високого рівня, яка призначена для представлення не тільки квантових операцій, але й класичної логіки, яка завжди буде необхідною. Потім компілятор переведе ці алгоритми в набір інструкцій, який можна виконати на квантовому комп'ютері. Подібно до класичних комп'ютерів, код, згенерований компілятором, знаходиться на рівні збірки, а асемблер, який ми розширили для цієї мети, називається Quantum Assembler (QASM), який є квантовою мовою ассемблера, який спочатку був визначений для відтворення квантових схем у [10]. Мікроархітектура забезпечить апаратну логіку керування, необхідну для виконання інструкцій на цільовій площині кубіта. Ці інструкції транслюються в мікроінструкції і через інтерфейсний шар відправляються в площину кубітів.

Всі ці процеси нам потрібно чітко розуміти, щоб ефективно розробляти та користуватися системами моделювання квантовтх обчислень.

РОЗДІЛ 1 ПРИНЦИПИ КВАНТОВОГО ОБЧИСЛЕННЯ

1.1 **Основні поняття**

Квантовий компьютер - Ми можемо думати про нього як про традиційний комп’ютер, який має доступ до квантової інформації та може маніпулювати нею. Інформація в звичайних ЕОМ або класичних комп’ютерах репрезентується через «0» і «1» на рівні електричного струму, що є бітами. Натомість квантова інформація використовує квантові біти або кубіти. Кубіт — це квантова система, яка має дві відмінні конфігурації, що відповідають бітовим значенням 0 і 1. В той же час у квантової системи є можливість застосовувати заплутанність: станом кубіта може бути будь-яка комбінація або суперпозиція двох конфігурацій. Так само рядок кубітів може бути в багатьох суперпозиціях своїх конфігурацій. Це дозволяє використовувати квантові ефекти і значно збагачує тип інформації, яку можна представити.

Нотація Дірака - дуже зручний спосіб вираження векторів, що використовуються в квантовій механіці. Це короткий вступ до «нотації в дужках» за теорією векторного простору в цілому. Далі наведено вступ, який зосереджується на основних визначеннях та векторних операціях це мова, яка створена для більш зручного вираження стану векторів квантовій механіці.

Позначення Бра-кета — базове позначення для опису квантових станів у теорії квантової механіки, що складається з кутових дужок і вертикальних смуг, буде далі показано на Рисунку 1.1. Його також можна використовувати для позначення абстрактних векторів і лінійних функціоналів в математиці. Таку назву вони отримали тому, що внутрішній

добуток (або добуток) двох станів позначається дужкою ⟨Φ|Ψ⟩ , що складається з лівої частини ⟨Φ| , називається бюстгальтером, і права частина, |Ψ⟩ , називається кет[7].

Внутрішній добуток, який також називають скалярним добутком або точковим добутком, обчислюється так само як матричний добуток бре і кета, що спрощується до добутку між вектором-рядком і вектором-стовпцем. Він записується в наступних формах, причому крапка (•) позначає скаляр:

$$\langle x| \cdot |y\rangle = \langle x||y\rangle = \langle x|y\rangle.$$

1.1.1 **Кубіт**

Квантовий біт або «кубіт» є квантовим аналогом класичного біта. Кубіт — це дворівнева квантово-механічна система. Класичний біт може бути тільки

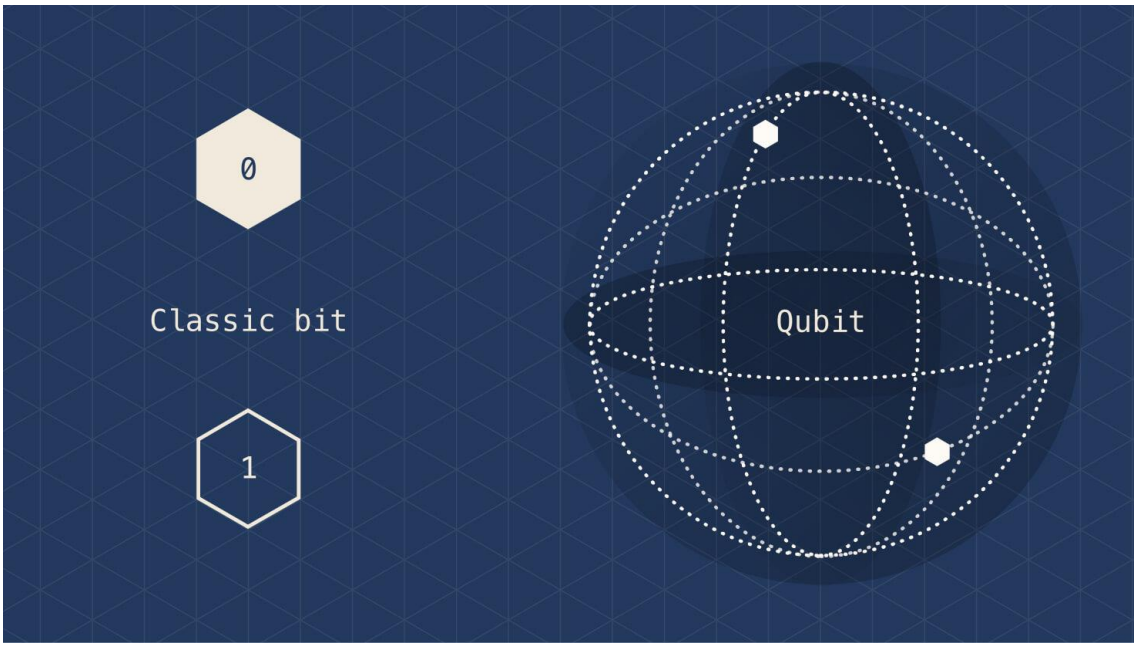


Рисунок 1.1 – Візуальне зображення кубіта [9]

в 0 або 1 станах. Однак квантова механіка має теорія, що стверждує наступне: кубіт може перебувати в суперпозиції обох станів одночасно, ця унікальна та комплексна властивість є фундаментальною для квантових

обчислень. Виходячи з цього, стан незплутаного кубіта можна представити у вигляді лінійної комбінації базового стану $|0\rangle$ і $|1\rangle$:

Вираження в виді формули (1):

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \equiv \begin{bmatrix} \alpha \\ \beta \end{bmatrix}; \quad |\alpha|^2 + |\beta|^2 = 1$$

1.1.2 Квантові входи та схеми

Квантовий алгоритм можна описати як схему, що складається з послідовності квантових операцій, що застосовуються до кубітів для проявлення нових квантових станів. Операції з вентилями — це лінійні перетворення, які застосовуються до квантового стану і можуть бути представлені у вигляді унітарних матриць.

З квантової теорії ми знаємо, що ідеальний квантовий комп'ютер має змогу моделювати будь-яку машину Тьюринга [23] і в теорії навіть всі локальну квантову систему [24]. Набір вентилів називається універсальним, якщо їх можна використовувати для створення квантової схеми, яка може наблизити будь-яку унітарну операцію з довільною точністю.[10]

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \quad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{2}$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3}$$

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4}$$

Було доведено, що будь-яку унітарну операцію можна апроксимувати з довільною точністю, використовуючи лише одиночні кубітні вентилі, такі як дані в рівняннях (2)(3), і вентиль CNOT, як зазначено в рівнянні (4) [10].

На рисунку 1.2 ми можемо бачити відому схему телепортації, де стан q0-кубіту телепортується в q2 за допомогою створення заплутаного стану між q1 і q2 і між q0 і q1. Після визначення як q0, так і q1, стан q2 міститиме стан q0, після застосування, коли необхідно, вентиля X або Z залежно від результату вимірювання.

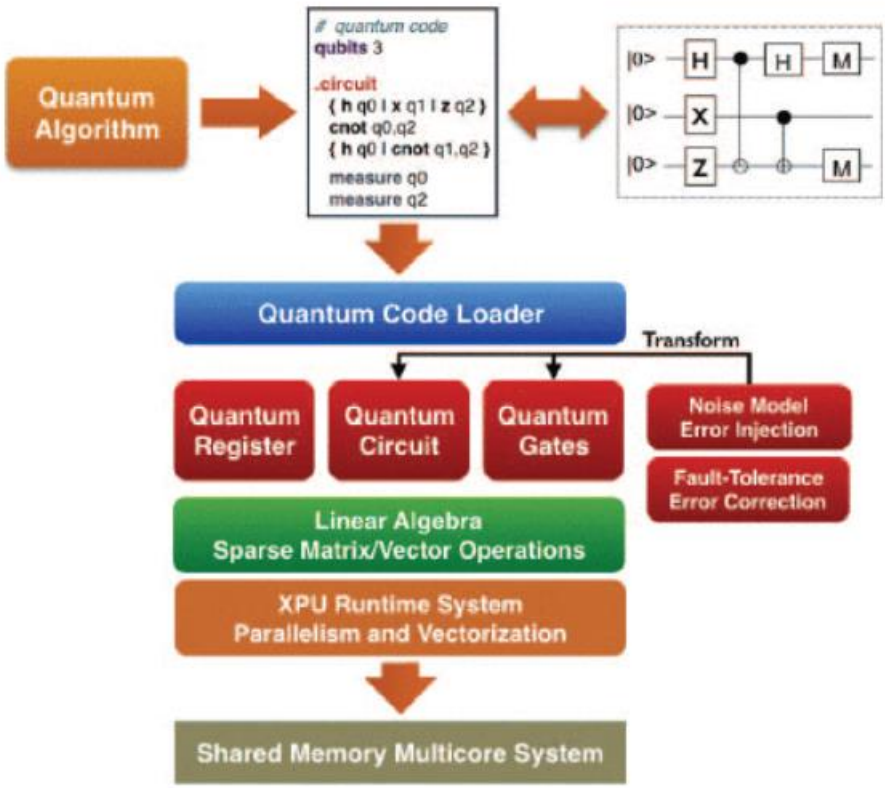


Рисунок 1.2 – Огляд конструкції симулятора квантових обчислень[11]

Треба мати на увазі, що паралельні квантові вентилі, що працюють на різних кубітах, можна зв’язати або об’єднати, щоб утворити більшу унітарну матрицю за допомогою тензорних добутків. У рівнянні (5) два вентиля Адамара об’єднуються, щоб утворити один двокубітний вентиль.

Послідовність вентилів, що працюють на одному і тому ж кубіті, можна об’єднати, перемноживши їх унітарні матриці.[12]

$$H \otimes H = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{5}$$

Об’єднання паралельних вентилів через тензорні продукти може генерувати величезні матриці, для яких мають бути величезні об’єми пам’яті. При об’єднанні більшої кількості воріт щільність матриць буде збільшуватися.

1.1.3 **Квантова Заплутаність**

Квантова заплутаність — це здатність квантових частинок порівнюваи результати своїх спінів один з одним. Коли кубіти заплутані, вони можуть утворюють зв’язану систему і передавати інформацію один одному. Ми можемо використовувати вимірювання з одного кубіта, щоб визначити спін, а самі інформацію про інший спін, приклад математичної викладки процесу наведено на рисунку 1.3.

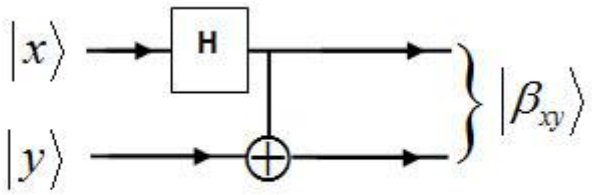


Рисунок 1.3 – Зображення принципу роботи квантових обчислень[12]

Маючи велику кількість заплутаних кубітів комп’ютери можуть обчислювати два в степені більше інформації ніж класичні компютери.[6]

1.1.4 **Квантова інтерференція**

Квантова інтерференція — це поведінка елементароної частки через суперпозицію, яка впливає на ймовірність його стану між часкою та хвилею тим чи іншим шляхом. Квантові сисмтеми обчислення спроектовані та створені, щоб зменшити максимально перешкоди в обчислюванні великих масивів данних та забезпечити найшвидші рішення. З цією метою Microsoft використовує топологічні кубіти, які стабілізуються шляхом маніпулювання їх структурою та оточування їх хімічними сполуками, які захищають їх від зовнішнього втручання[6].

1.1.5 **Надпровідник**

Надпровідник – це такий провідник, що при таких наднизьких температурах може показувати характеристики матеріалу, що не має опору при проходженні струму. Це робить їх «надпровідниками». Коли електрони проходять через надпровідники, вони збігаються, утворюючи «куперівські пари». Такі пари переносять заряд через квантовий бар’єри або ізолятори за допомогою процесу, який ми знаємо з квантової фізики як тунелювання. Два надпровідники, розміщені по обидва боки ізолятора, утворюють джозефсонівський перехід.

1.1.6 **Суперпозиція**

Сама по собі елементарна стабільна частка не може бути придатною до обчислення. Але вона може мати важливу характеристику: перевести квантову інформацію, яку він зберігає, у стан суперпозиції, що, представляє собою комбінацію всіх можливих конфігурацій кубіта, рисунок 1.3.

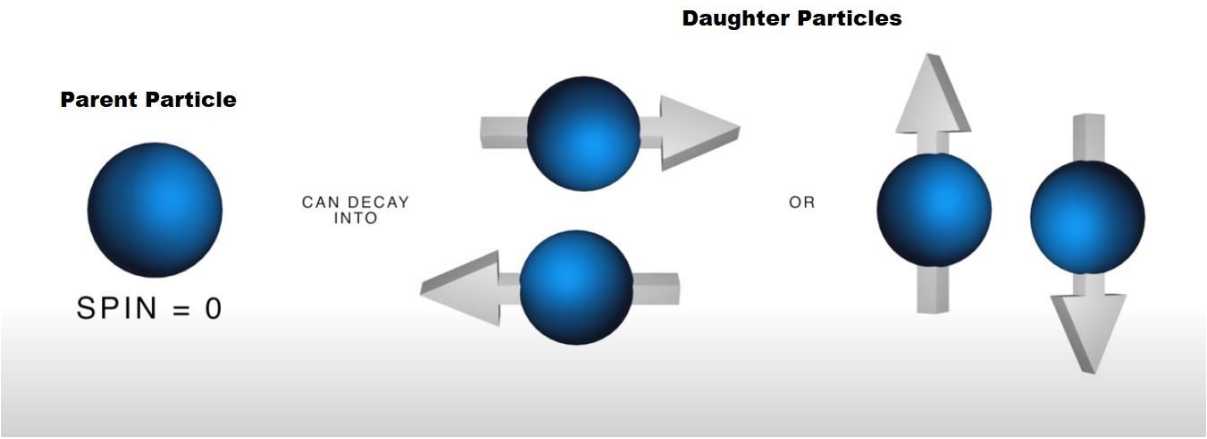


Рисунок 1.3 – Візуалізація квантової заплутанності [13]

Деяка кількість кубітів у суперпозиції можуть створювати складні багатовимірні обчислювальні простори. У таких видах простору можна по новому побачити підхід до обчислення.

1.2 **Модель та алгоритм кватового обчислення**

Після ознайомлення з основними поняттями та розбором тем які допоможуть нам зрозуміти природу та логіку квантових обчислень ми можемо розглянути схему на якій опишемо сам процес та всі його етапи.

Квантові обчислення вимагають фізичного маніпулювання — ініціалізації, контролю та вимірювання стану багатьох кубітів. Керування такими явищами виконується за допомогою ЕОМ. Це призводить потенціальнох доступності квантових обчислень, які використовують квантові системи для виконання більш ефективних обчислень. Але здавалося, що тема все ще представляла насамперед академічний інтерес.

Далі буде розібрано, чому ми можемо вивести цю теорію на наступний рівень, коли вона вже буде мати практичний потенціал та великі перспективи в цілому. І ми зможемо розібратися в цих процесах чікіше за допомогою ілючтарції 1.4 та поясненням.

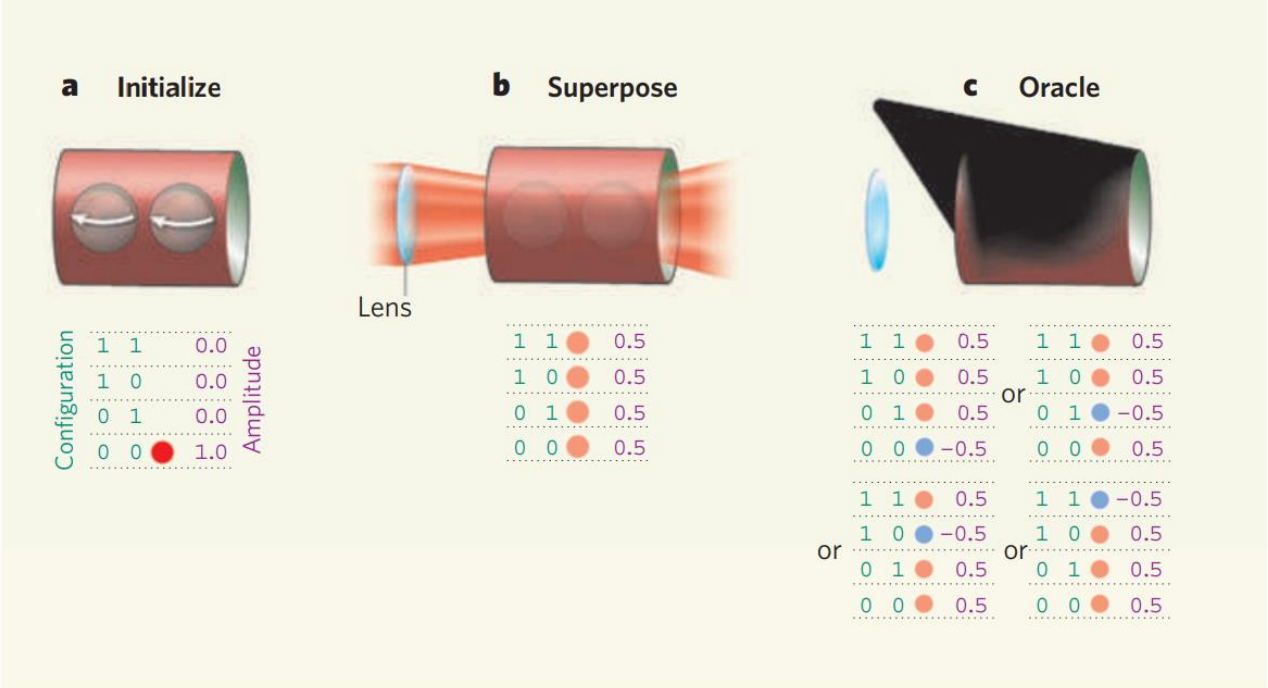


Рисунок 1.4 – графічне зображення алгоритму обрахувань[14]

Етапи обчислень:

А. (Ініціалізація) Показаний розрахунок вимагаєдва кубіти, зображені у вигляді частинок (сині сфери в мідному корпусі) спіна ½. Частка конфігурації, що обертаються вліво або праворуч, відповідають бітовим значенням 0 і 1 відповідно. Сильна взаємодія (не показано) спонукає кубіти крутяться вліво (позначені стрілкою), які є потім ізольовані від декогерентності процесів у мідному корпусі. Ми може візуалізувати стан двох кубітів як хвильова функція на просторічотири можливі конфігурації — 00, 01, 10 і 11. Кожна конфігурація сприяє амплітуді хвильова функція. Початковий стан має всю свою амплітуду на конфігурація 00 (червоне коло).

В. (Суперпозиція) Обчислення ми починаємо базової операції (показана тут як світло, що проходить крізь лінзу і сяє на частинки), яка змінює стан на рівномірну суперпозицію конфігурацій. Нова хвильова функція має рівні амплітуди в кожній конфігурації. Якщо ми виміряємо стан (що ми поки не повинні робити), то ймовірність спостереження певної

конфігурації визначається її амплітудою в квадраті (0,52 = 0,25 у цьому випадку). Відтінок і колір кіл, пов'язаних з кожною конфігурацією, представляють величину і фазу (червоний для позитивної фази, синій для негативної) амплітуд. Хоча візуалізація хвильової функції можлива для будь-якої кількості кубітів, вона стає надзвичайно громіздкою через експоненційне зростання кількості можливих конфігурацій.

C. (Оракул) Ми застосовуємо обчислення «чорного ящика», або оракула, поведінку якого важко передбачити, якщо не використовувати його. Оскільки чорний ящик діє за один крок на стан, який включає всі конфігурації одночасно, він використовує «квантовий паралелізм» (не плутати з класичним паралелізмом). У цьому випадку поведінка чорного ящика полягає в тому, щоб «позначити» одну конфігурацію шляхом зміни фази її амплітуди. Але який із них змінив? На цьому етапі вимірювання конфігурації нічого не виявить, оскільки ймовірності результатів вимірювання залежать лише від величин. [15]

D. (Інтерференція) Далі ми використовуємо перешкоди, щоб зосередити амплітуди на позначеній конфігурації. З точки зору хвильових функцій, це включає рекомбінацію амплітуд, ніби шляхом їх передачі через інтерферометр. Тут діє інший тип світла (синій).

E. (Вимірювання) Стан кубітів вимірюється шляхом відкриття ізоляційного корпусу та перевірки орієнтації спінів. Оскільки лише одна амплітуда відмінна від нуля, результат є детермінованим і розкриває розташування позначки. У цьому випадку він виявляється на рівні 10. Для багатьох алгоритмів існує багато відмінних від нуля амплітуд, і результати вимірювання є імовірнісними. Для цього квантового обчислення було

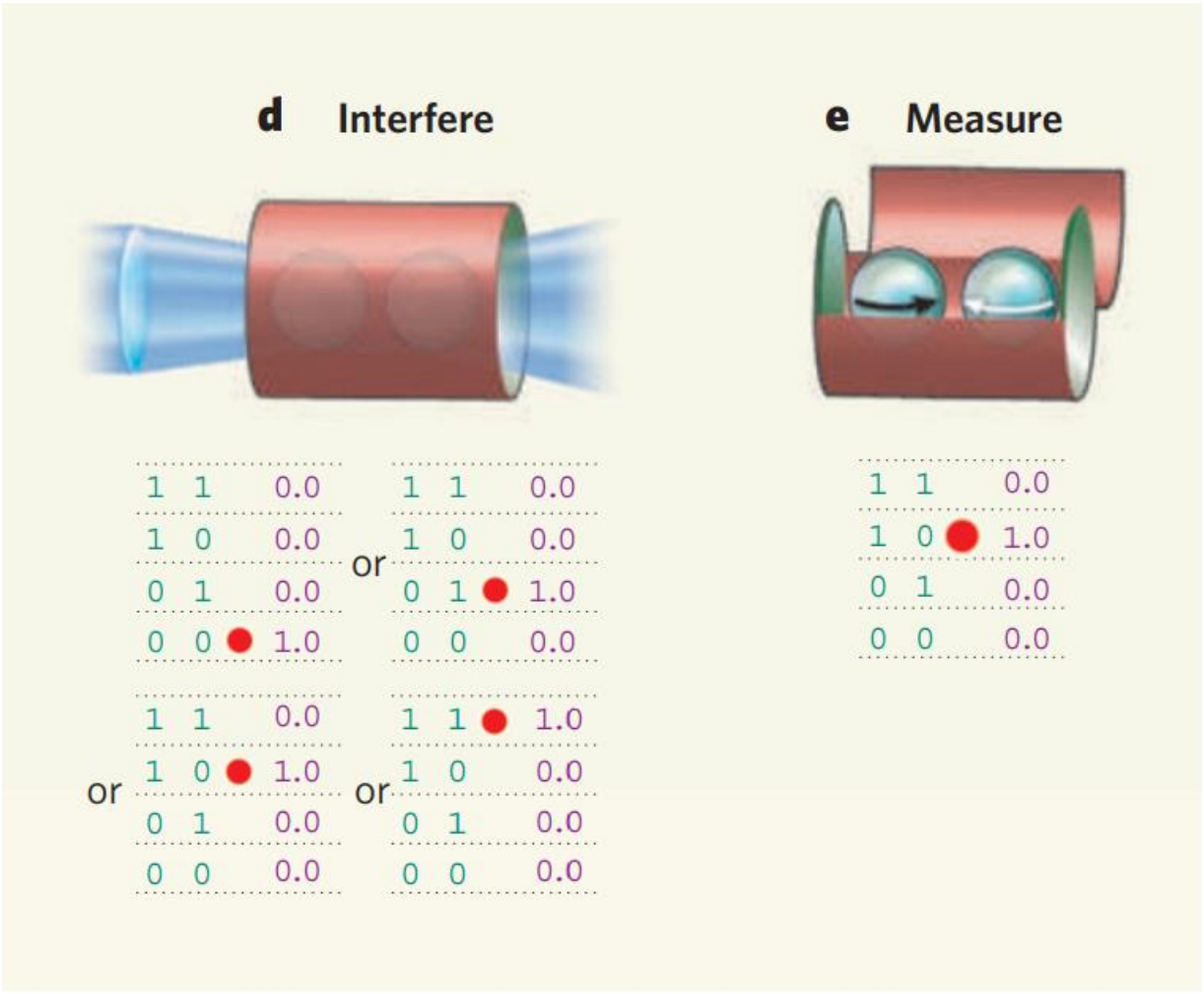


Рисунок 1.5 – Ілюстрація різних рівні програмування квантових алгоритмів[14]

потрібно одне застосування чорного ящика. Для аналогічних класичних обчислень знадобиться до трьох застосувань.

РОЗДІЛ 2 ОГЛЯД ДИЗАЙНУ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ СИМУЛЯТОРІВ

Як і в класичних комп'ютерах, необхідно мати спосіб для надсилання інструкцій і отримання результатів від нашої квантової системи, іншими словами мова для прогрмаування його. Існують різні рівні мов програмування з якими ми вже познайомилися: починаючи вім мови асемблера (також відомих як мови інструкцій квантової машини), які надчилають директивні комнди процессору, до мов програмування високого рівня, де квантові алгоритми вже встановлені просо вимулюються, і нам потрібно просто користуючись більш легким середовищем вводити основні задачі для обчислення нашої задачі.[16]

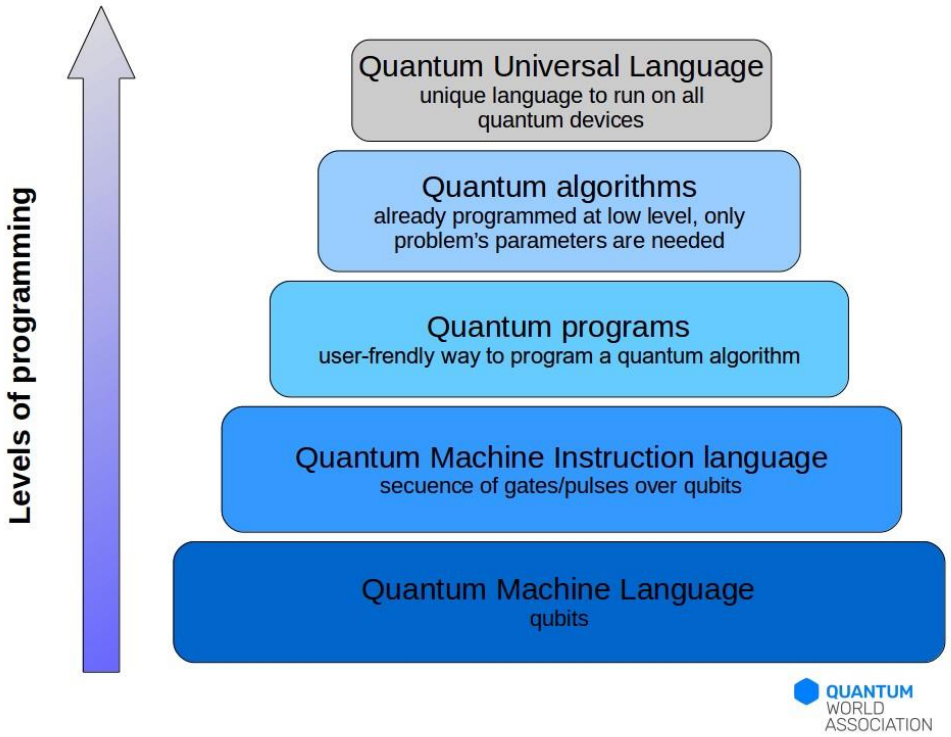


Рисунок 2.1 – Рівні програмування квантової машини [6]

Квантовий комп’ютер в реальності є гібридною системою, яка складається з квантового пристрою (апаратного забезпечення) і компонента зіичайного комп’ютера, що відсилає інструкції апаратному забезпеченню та отримує і обробляє результати (програмне забезпечення, далі ПО). Саме цьому основному ПО відповідають квантові мови. Деякі з них є належними мовами програмування, як і класичні. Інші фактично є бібліотеками, запрограмованими з використанням якоїсь добре відомої мови (Python, C++), що дають можливість користувачу легше оперувати процессами квантової машини.[17]

Важко побачити чіткі парелелі між звичайною ЕОМ і квантовими рівнями програмування. Ми спробуємо класифікувати деякі з найбільш відомих квантових мов програмування на різних рівнях абстракції, особливо ті, які розробляються техічними компаниями та невеликими стартапами. Звичайно зробити вечерпуваний перелік не вдасться. Як і у випадку з ландшафтом квантових обчислень, представленим у нашій першій статті про квантові обчислення, список з кожним днем збільшується.

2.1 QX симулятор

2.1.1 Огляд архітектури

Як показано на рисунку 2.1, архітектура симулятора QX організована за модульною системою, що дозволяє легко вводити майбутні розширення. Компоненти:

Завантажувач квантового коду, який зчитує вхідний квантовий код і створює відповідну виконувану схему. Він також може завантажувати параметри виконання, такі як модель помилки для використання та схема планування квантових воріт.

Quantum Core має в собі основні елементи моделювання: регістр кубітів і квантові схеми.

Моделі помилок вводять дискретні помилки в квантову схему або змінюють квантові вентилі для імітації шумного виконання.

Модуль відмовостійкості створює відмовостійку схему на основі вхідної несправної ланцюга. На цей день модуль підтримує код Steane [25] і незабаром буде підтримувати код більш високого рівня. Детальне обговорення цього модуля не надається через обмеженість простору.

```
qubits 3 # define quantum register of 3 qubits

.init   # first sub-circuit
   x q0
   display
.bell_pair  # create an EPR pair
   h q1
   cnot q1,q2
   display
.encode
   cnot q0,q1
   h q1
   measure q0
   measure q1
   display
.decode
   cx b1,q2  # binary-controlled X gate
   cz b0,q2  # binary-controlled Z gate
   display
```

Рисунок 2.2 – Схема квантової телепортації [18]

- Рівень лінійної алгебри реалізує розріджені матричні/векторні операції, необхідні для моделювання, зосереджуючись на ефективному використанні розріджених операторів.

2.1.2 Код квантової збірки

А. Створення кубітів

1) Квантовий регістр

Квантовий регістр - це деяка кількість кубітів, які є аналогом класичного регістру процесора. Вони містять квантовий стан, що сам має квантові амплітуди різних обчислювальних базових станів. Розробник імплементує розмір квантового регістра або данні по кубітам, як у прикладі, показаному в листингу 2.2, що визначає квантовий регістр із 3 кубітів.[19]

2) Реєстр вимірювань

Коли визначено квантовий регістр, регістр вимірювання (MR) неявно створюється і пов'язується з квантовим регістром. MR — це класичний регістр, який містить результати вимірювань різних кубітів. Ці результати можна використовувати для контролю квантових операцій. Біти MR за замовчуванням іменуються відповідно b0, b1… і пов'язані з результатами вимірювання кубітів q0, q1… тощо.[20]

B. Квантовий контур

Квантові алгоритми можна репрезентувати у вигляді квантової схеми, що визначають різні операції над кубітами та їх взаємодію. Схема на основі квантів частіше складається з чистих квантових гейтів чи воріт, таких як ворота Адамара, CNOT і Паулі. Ми зазначаємо, що наш QASM забезпечує більшість поширених однокубітних, двокубітних і трьохкубітних вентилів. Треба звернти увагу на те що є вимірювання стану кубіта, що створює класичний біт. Що далі може бути результатом квантових обчислень або може використовуватися для умовного виконання додаткових квантових вентилів через двійкові керовані вентилі, наприклад у схемах виправлення помилок.

У листингі 2.2 показана схема телепортації (показана на малюнку 2.2), написана в QASM. Схема розділена на чотири функціональні підсхеми під назвою «init», «bell_pair», «encode» і «decode» і включає в себе квантові вентилі, двійкові керовані вентилі та вимірювання. У наступних параграфах ми представляємо декілька функцій QASM.

```
1  # define a quantum register of 7 qubits
2  qubits 7
3
4  .init  # initialization sub-circuit
5      measure q3 # measurement outcome in b3
6      cx b3,q3   # binary-controlled x gate using b3
7      {prepz q0 | prepz q1 | prepz q2}  # parallel gates
8      {x q3 | h q0 | h q1 | h q2 }
9      h q3
10 .grover(2) # 2 iterations loop
11     { x q2 | toffoli q0,q1,q4 }
12     toffoli q1,q4,q5
13     ...
14     { x q2 | h q0 | h q1 }
15     h q2
16 .result # measurement
17     { h q3 | measure q0 | measure q1 | measure q2 }
18     measure q3
19     display
```

Приклад алгоритму Гровера [18]

C. Підтримка петлі

Структура управління циклом є важливим компонентом багатьох квантових алгоритмів, таких як алгоритм Гровера. Наразі QASM забезпечує підтримку циклу FOR, просто додаючи кількість разів, коли підсхема повинна бути виконана відразу після назви підсхеми, наприклад у рядку 10 прикладу алгоритму Гровера, наведеного в листингу 2.[21]

D. Планування квантових воріт

Перш ніж створити остаточний виконуваний файл, будь-який компілятор змінить порядок інструкцій для досягнення найкращої продуктивності з точки зору використання ресурсів і часу обчислень. Те ж саме стосується квантових схем, де також намагаються зменшити загальний

час виконання та використовувати кубіти найбільш економним способом. Як для моделювання, так і для реальних експериментів важливо зрозуміти, як квантовий шум впливає на затримку ланцюга.

QX дозволяє експериментувати з різними моделями шуму та параметрами часу для вивчення, наприклад, стійкість схеми до шумів і те, як планування може покращити цю стійкість. Планувальник також повинен знати про можливі апаратні обмеження, такі як кількість вентилів, які можуть виконуватися одночасно, і про можливий компроміс між швидкістю виконання та декогерентністю кубітів. QASM дозволяє виразити паралелізм між квантовими воротами. Ворота, укладені в дужки, виконуються паралельно, як у прикладі схеми Гровера, наведеному в листингу 2.

E. Відображення кубітів

Важливою відмінністю від класичних обчислень є те, що кубіти відображаються на 2D-решітці і що логіка застосовується до кубітів за допомогою квантових вентилів. Сучасні квантові технології вимагають, щоб кубіти розташовувалися поруч один з одним при застосуванні мультикубітного вентиля. Переміщення кубітів виконується за допомогою дорогих ланцюжкових операцій підкачки, тому відображення кубітів на 2D-решітці є критичним для мінімізації переміщень кубітів і покращення локалізації даних.

QASM пропонує набір мета-інструкцій для розміщення або відображення кубіта в задану позицію на площині кубіта, наприклад, карта q0, 3, 4, де q0 відображається на позиції з координатами x=3 і y=4. Після початкового відображення кубіти можуть бути явно переміщені з однієї позиції в іншу за допомогою елементів «переміни місцями» за найкоротшим шляхом на решітці. [22] Наприклад, mov q0,5,1 переміщує q0 до нових (x, y)

координат. Ця функція спрямована на полегшення дослідження оптимального відображення кубітів і маршрутизації кубітів.

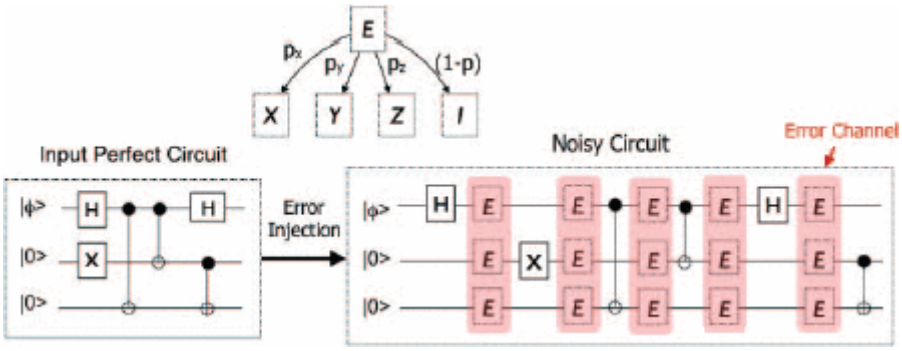


Рисунок 2.3 – Реалізація симетричного деполяризаційного каналу в QX. [29]

Деполяризаційний канал

Квантовий деполяризуючий канал є стандартною моделлю для шуму в квантових системах. Для d-вимірної системи деполяризуючий канал відображає кубіт, спочатку в стані ρ, у лінійну комбінацію самого стану та повного змішаного стану Id, де I — ідентична матриця, а p — ймовірність деполяризації кубіта.

$$\triangle_p(\rho) = (1-p)\rho + \frac{p}{d}I \qquad (6)$$

Канал деполяризації може бути визначений рівнянням 7, де px,py і pz – відповідні ймовірності операторів перевороту бітів Паулі, перемикання фази та комбінованого перевороту фази-біту, що діють на деполяризований

кубіт. Коли px,py і pz рівні, то деполяризуючий канал називається

$$\rho \to (1-p)\rho + p_x X\rho X + p_y Y\rho Y + p_z Z\rho Z \qquad (7)$$

симетричним.

У симуляторі QX деполяризаційний канал реалізується шляхом введення дискретних похибок під час різних кроків ланцюга у вигляді квантових затворів Паулі з ймовірністю помилки p. Введена помилка може бути помилкою перевороту бітів або помилки перемикання фази або їх комбінацією. Рисунку 4 ілюструє цей процес ін'єкції помилок: ідеальна схема перетворюється на зашумлену ланцюг шляхом ін'єкції помилки на кожному кроці часу.

Оптимізація QX і експериментальні результати

У цьому розділі ми обговорюємо оптимізації та порівнюємо QX з іншим найсучаснішим симулятором квантового комп'ютера, а саме LIQUi|⟩ [18]. Ми пояснюємо, як QX порівнює швидкість моделювання, використовуючи різні контрольні показники. Наша експериментальна апаратна установка базується на платформі Linux SMP з двома процесорами Intel(R) Xeon(R) E5-2683 з 396 ГБ пам'яті. [24]

A. Шлях для одного кубіта

Як ми бачили в розділі III, застосування одного кубітного вентиля до вектора квантового стану відповідає множенню цього комплексного вектора на тензорний добуток матриці вентилів (застосований до цільового кубіту) та ідентичних матриць (застосованих до решти кубіти). Результуюча тензорна матриця дуже розріджена, і тому дозволяє провести ряд оптимізацій для зменшення споживання пам'яті та кількості операцій. У наступних параграфах ми описуємо, як ми можемо ефективно реалізувати

основну складну арифметику, а потім як поступово зменшувати операції з плаваючою комою, щоб прискорити виконання.

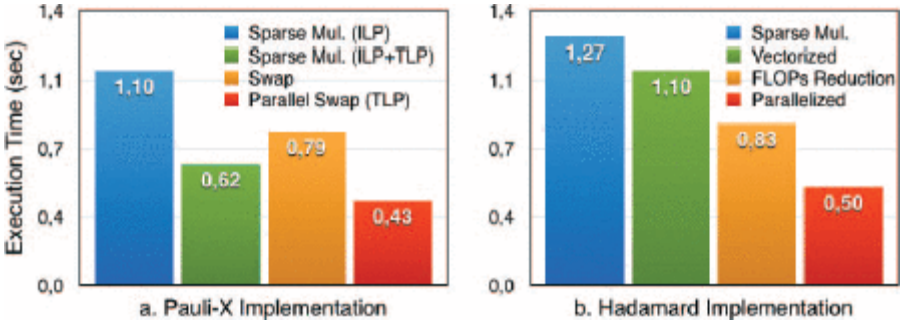


Рисунок 2.4 – (a) Вплив різних оптимізацій на час виконання одного кубітного вентиля. (b) Продуктивність pauli-X на основі заміни порівнюється з реалізацією на основі множення. [29]

1) Паралелізм на рівні інструкції

Перша оптимізація базової версії коду пов'язана з векторизацією складної арифметики. Сучасні процесори забезпечують кілька векторних наборів інструкцій, таких як SSE, AVX і FMA, що дозволяють виконувати операції з подвійною точністю 2 і 4 способами. Зокрема, SSE3 надає спеціальні інструкції для ефективного прискорення складної арифметики. Ми використовуємо більш пізній набір інструкцій у поєднанні з розширенням FMA, яке дозволяє виконувати операції множення-додавання, цей набір інструкцій використовується для ефективної реалізації розрідженого векторно-матричного множення. Реалізація з ручною векторизацією досягла 24% збільшення часу виконання в порівнянні з початковою реалізацією, автоматично векторизованою компілятором GNU. [26]

2) Скорочення операцій з плаваючою комою

Як показано в розділі III цієї статті, матриці загальних квантових вентилів відомі під час компіляції, що дозволяє проводити більш конкретні оптимізації. Наприклад, елементи матриці Адамара є чистими дійсними комплексними числами з нульовими уявними компонентами, що дозволяють зменшити кількість операцій, породжених комплексним множенням, від 6 до 2 операцій подвійної точності. Як показано в рівняннях 4 і 5, деякі інші матриці, такі як матриці фази або матриці Паулі-Гейтса, є діагональними або протидіагональними матрицями, що дозволяють ще менш складні множення. Більше того, застосування квантових вентилів в основному складається з послідовного комплексного множення та додавання, що дозволяє нам виконувати подальшу оптимізацію шляхом об'єднання комплексного множення та додавання для досягнення високої продуктивності та меншої кількості доступів до пам'яті. Остання реалізація скоротила загальний одноядерний час виконання операції шлюза Адамара приблизно на 39% порівняно з базовою послідовною реалізацією. Вплив різних кроків оптимізації на продуктивність одного кубітного вентиля в схемі з 28 кубітами зображено на малюнку 5.

3) Реалізація на основі свопів

Гейт Pauli-X — це звичайний однокубітний вентиль, який виконує переворот бітів на цільовому кубіті. Якщо ми звернемося до рівняння 3, ми можемо помітити, що унітарна матриця воріт Паулі-X є матрицею перестановок, отже, тензорний добуток цієї матриці з ідентичною матрицею все ще є матрицею перестановок. Однією цікавою властивістю цього типу матриці є те, що при множенні на вектор вона просто переставляє елементи цього вектора, отже, множення такої матриці на наш вектор квантового стану переставляє скінченну кількість її елементів. Тому дороге комплексне множення можна замінити швидшими операціями обміну, які зменшують час виконання шлюза на 40%, як показано на малюнку 5.

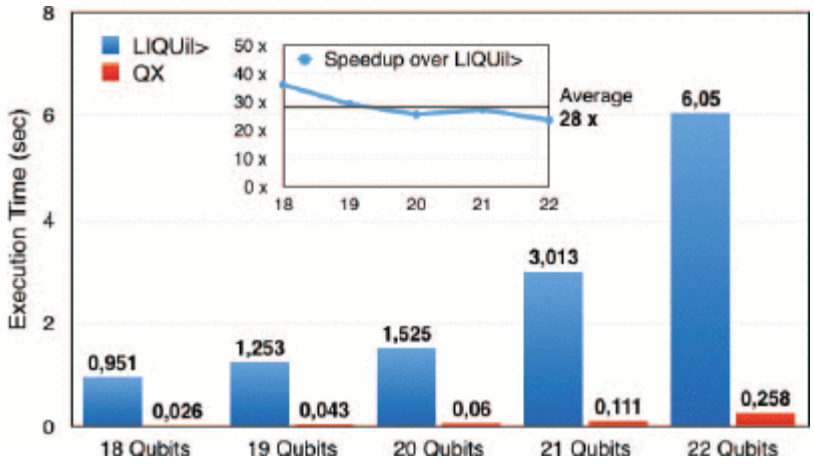


Рисунок 2.5 – Порівняння продуктивності симуляторів QX і LIQUi, які виконують схему QFT на різній кількості кубітів. [29]

4) Паралелізм на рівні потоків

Паралелізм на рівні потоків має вирішальне значення для досягнення хорошої масштабованості в багатоядерних системах. Ми паралелізуємо множення векторної матриці та операції підкачки на рівні потоку, використовуючи конструкцію parallel_for фреймворку паралельного програмування XPU [27]. Середовище виконання XPU динамічно виявляє доступні процесорні блоки та розподіляє навантаження на ці блоки. На малюнку 5 показано вплив паралельності 4 потоків на продуктивність одиночного кубітного вентиля, що зменшує вдвічі час виконання порівняно з розрідженим множенням.

B. Керовані ворота

Керовані однокубітні вентилі зменшують кількість операцій вдвічі для кожного контрольного кубіта. Наприклад, вентилі CNOT і Toffoli є відповідно одиничними і двома кубітними керованими вентилями Паулі-X, які широко використовуються в багатьох квантових алгоритмах. У їхніх реалізаціях на основі підкачки кількість операцій підкачки може бути

зменшена відповідно у 2 та 4 рази і таким чином можна скоротити час моделювання.

C. Ефективність моделювання квантових алгоритмів

У наступних параграфах ми порівнюємо продуктивність нашого симулятора з симулятором LIQUi|⟩ за допомогою набору популярних квантових алгоритмів.

1) Квантове перетворення Фур'є

Квантове перетворення Фур'є (QFT) є фундаментальною частиною багатьох квантових алгоритмів, таких як алгоритм факторингу Шора [3] або алгоритм оцінки квантової фази. QFT в основному складається з вентилів Адамара і контрольованих фазових зсувів, за якими слідує серія змін кубітів, що повертають кубіти. На малюнку 6 показана досягнута продуктивність QX-симулятора в порівнянні з симулятором LIQUi|⟩, наш симулятор досягає в середньому прискорення в 14 разів.

2) Контур заплутування

Квантова заплутаність — це фізичне явище, яке виникає, коли пари кубітів взаємодіють таким чином, що квантовий стан кожного кубіта неможливо описати незалежно. Контур заплутаності є загальноприйнятою схемою в квантовій інформації і складається з вентиля Адамара і набору вентилів CNOT, за якими слід вимірювання кубітів. Ми порівнюємо продуктивність моделювання QX цієї схеми з симулятором LIQUi|⟩ для різної кількості кубітів. Як показано на малюнку 7, QX перевершує симулятор LIQUi|⟩ і досягає в середньому 46-кратного прискорення над ним. Зауважимо, що симулятор LIQUi|⟩ компілює та оптимізує схему перед її

виконанням, вартість компіляції та оптимізації включаються в загальний час виконання.

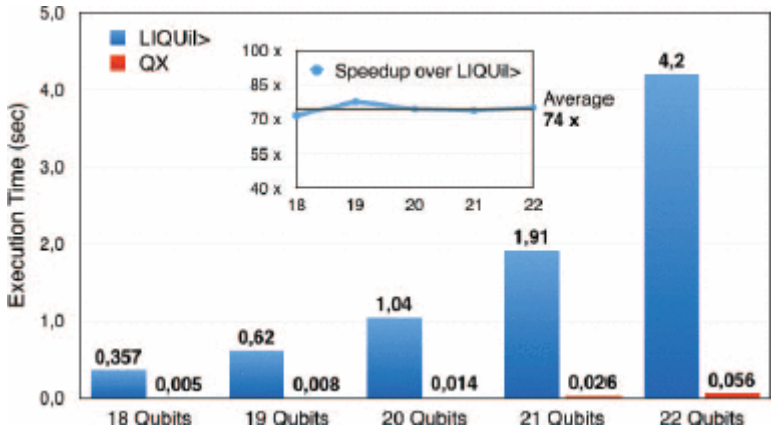


Рисунок 2.6 – Порівняння часів моделювання схеми заплутування QX-симулятором та симулятором LIQUi|⟩. [29]

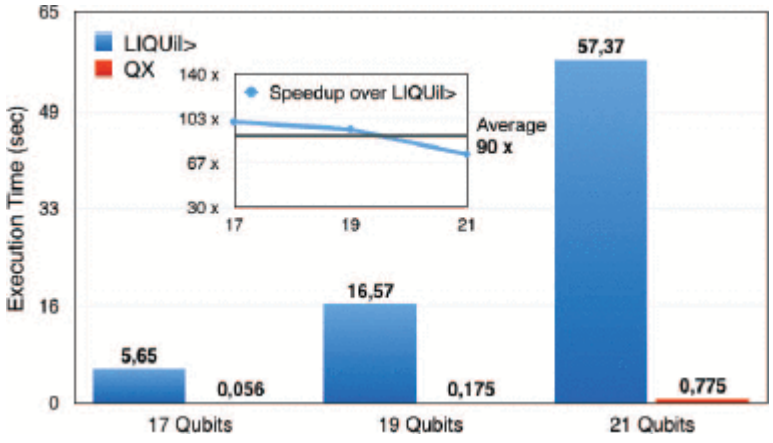


Рисунок 2.7 – Порівняння часу моделювання схеми гровера за допомогою QX-симулятора та симулятора LIQUi|⟩

3) Алгоритм Гровера

Ми реалізуємо алгоритм пошуку Гровера для різних розмірів задач (17,19 і 21 кубіт). Моделювання схеми з 21 кубітом вимагає виконання 2388 вентилів, включаючи вентилі H, X, CNOT і Toffoli. Ми пишемо відповідну

схему, використовуючи код ассемблера QX і мову F#, яку використовує LIQUi|⟩. Ми порівнюємо продуктивність двох середовищ моделювання, малюнок 8 показує, що QX-симулятор перевершує LIQUi|⟩, досягаючи значного прискорення в 95 разів для схеми з 19 кубітами.

2.2 Порівняння різних симуляторів

Ми зосередимося на квантових мовах, які використовуються компаніями, які розробляють квантові комп'ютери. Не всі з них використовують цю стратегію з відкритим кодом для створення своїх кодів, тому мова, яку використовують деякі компанії, не є загальнодоступною, навіть назва. Почнемо з компаній та тим, що вони роблять, а після цього роздивимость кожен з їх продуктів конретніше ті зрівняємо їх.

- IBM: вони розробили Quantum Information Software Kit (QISKit), який є повноцінною бібліотекою для написання, моделювання та виконання квантових програм. Нещодавно вони розділили його на чотири частини: Terra, яка дозволяє програмувати на рівні квантових вентилів та імпульсів (квантові ворота реалізуються за допомогою послідовностей імпульсів); Aqua, більш високий рівень програмування для запуску алгоритмів, що використовуються в квантовій хімії, оптимізаційних задачах та ШІ; Ignis, щоб охарактеризувати помилки та покращити реалізацію воріт; Aer, для вивчення меж квантових обчислень за допомогою моделювання в класичних пристроях. QISKit перекладає квантові програми на мову нижчого рівня під назвою QASM, яка є його мовою квантових інструкцій. Qiskit є фаворитом порівняно з іншими мовами, бо він має деякі переваги. Цю мову бібліотеку можна доволі легко вивчити і розібратися в концепціях представлених в документації, так як це звичайна бібліотека, то впевненим користувачам Python може бути дуже

зручно починати вивчення мови заме з цього програмного забезпечення. Більш того,

- Rigetti Computing: так само вони створили Forest, середовище розробника для написання та виконання квантових програм. Їхні квантові пристрої програмуються за допомогою бібліотеки Python під назвою pyquil, яка перекладає програми, написані в термінах квантових воріт, на мову нижнього рівня, яка називається quil. Вони також розробили бібліотеку Grove, яка включає в себе такі алгоритми, як Variation Quantum Eigensolver, що використовується в квантовій хімії, або QAOA, що використовується для задач оптимізації. [28]

- Microsoft: ця компанія пропонує Quantum Development Kit, який містить квантовий симулятор, бібліотеки для реалізації квантових алгоритмів і повноцінну мову квантового програмування під назвою Q#, яка доступна як окремо завантажене розширення для Visual Studio. Microsoft все ще розробляє свій квантовий комп'ютер з топологічними кубітами, тому наразі вони пропонують квантовий симулятор для запуску квантових програм.

- D-Wave: їхнє програмне середовище містить qbsolv, квантову мову, яка допомагає користувачам відображати свою проблему QUBO (тип проблем, які потрібно вирішувати за допомогою своїх квантових відпалювачів) із підключенням кубітів пристроїв D-Wave та перекладає програму на її квант. мова навчання. Вони також пропонують кілька бібліотек вищого рівня для реалізації деяких проблем оптимізації, які називаються QSage і ToQ. Національна лабораторія Лос-Аламоса

розробила мову квантового макроассемблера під назвою QMASM, яка заповнює прогалини в програмній екосистемі D-Wave.

- Google: хоча не є офіційним продуктом Google, її група штучного інтелекту розробила Cirq, квантову мову, яка складається з бібліотеки Python для написання, маніпулювання та оптимізації схем і повторного запуску квантових комп’ютерів і симуляторів. Наразі він використовується деякими компаніями та групами для запуску квантових пристроїв Google. Існує бібліотека для запуску OpenFermion — бібліотека з відкритим вихідним кодом для отримання та керування уявленнями ферміонних систем, дуже корисна для квантової хімії, для моделювання на квантових комп’ютерах — запрограмована для кодів Cirq під назвою OpenFermion-Cirq. На початок 2021 року Cirq працює над пропозицією доступу до одного з реальних квантових комп'ютерів Google. Тим часом, все ще можна створювати алгоритми та схеми та тестувати їх на квантовому симуляторі.

- Google пропонує різні посібники, які допомагають новачкам перейти від нульового до експертного рівня квантової симуляції за допомогою Cirq.

- Xanadu: вони розробили Strawberry Fields, повну бібліотеку, спеціально націлену на квантові обчислення безперервної змінної, що є квантовим комп'ютером на основі фотонів, який вони проектують і конструюють. Квантові схеми написані за допомогою квантової мови програмування Blackbird, яка є мовою асемблера для їхніх квантових пристроїв. [29]

- Qilimanjaro: цей стартап запустив альфа-версію Qibo, мови, яка буде використовуватися для програмування його квантового відпалювача. Вони тільки почали конструювати квантовий пристрій, тому пропонують квантовий симулятор для запуску програм. Ця компанія також має на меті використовувати цю мову як квантову універсальну мову для будь-якого іншого сервера.

Як ви могли помітити, існує багато мов квантових обчислень. Кожна компанія робить пропозицію програмувати свої пристрої. Який з них буде кращим і нав'язаний над іншими? Хто знає. Якщо озирнутися назад, то це не нова дискусія. У нас подібна з класичними мовами програмування. Зрештою, вони страждають від природного відбору; виживають лише найуживаніші. Тим часом деякі люди намагаються спростити роботу дослідників, розробляючи мови, які перекладають квантові програми на вищезазначені в залежності від того, який пристрій ви хочете використовувати. Однією з цих квантових універсальних мов є ProjectQ, який можна використовувати для програмування пристроїв IBM і Google з

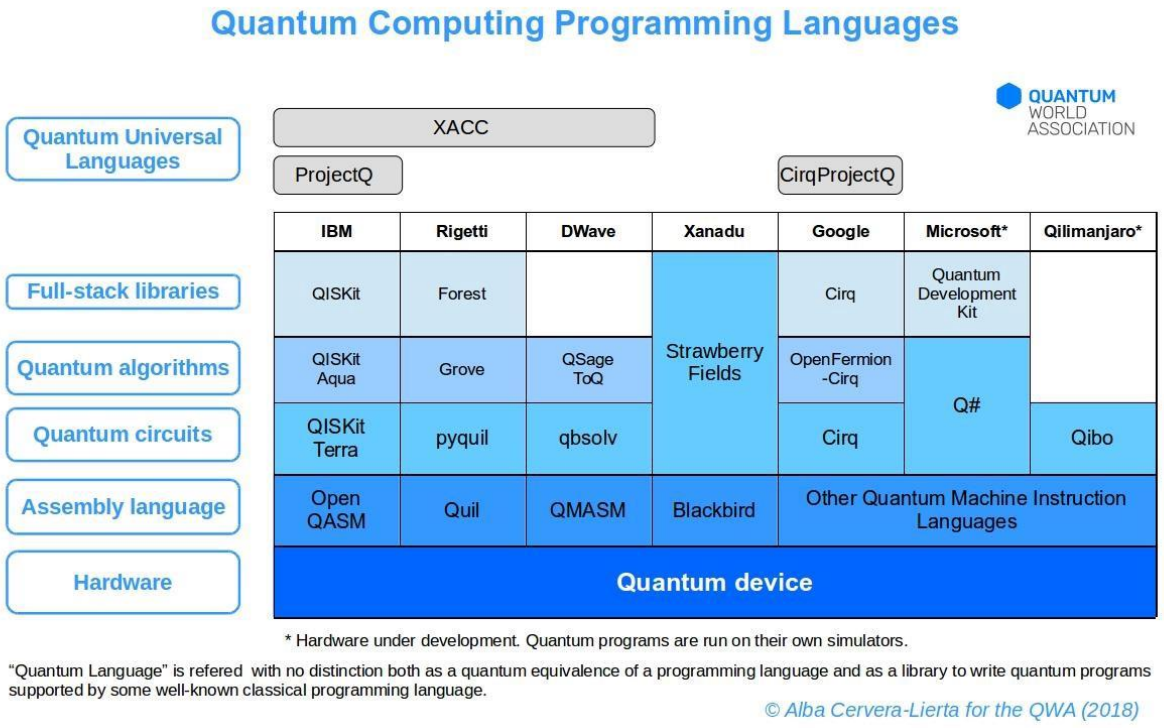


Рисунок 2.8 — Класифікація деяких з існуючих мов квантових обчислень за рівнем програмування. [30]

Його розширенням CirqProjectQ. Іншим є XACC, яка є повною бібліотекою для програмування квантових комп'ютерів IBM, Rigetti та D-Wave.

Квантові мови все ще розробляються, як і квантові пристрої. Вони розвиваються разом з апаратним забезпеченням, з яким їм доводиться спілкуватися. Як ви знаєте, у нас є кілька прототипів квантових комп'ютерів, які складаються з кількох шумних кубітів, але ми прагнемо створити відмово стійкі квантові комп'ютери, тобто квантові пристрої, які можуть виконувати складні алгоритми без помилок. У наступній статті Medium ми поговоримо про цю проблему: короткочасні шумні проти відмовостійких квантових обчислень.

Дослідивши всі можливі та доступні варіанти для симуляції квантових систем, ми можемо зробити висновок, що найкащим вибором може стати розробка IBM в виді бібліотеки для мови програмування Python. Наступним кроком для дослідження цього питання, ми маємо розглянути та вирішити деякі проблеми які будуть репрезетативні та відповідатимуть темі практики. В Наступному розділі ми розглянемо види проблем та алгоритмів для їх вирішення які нам пропонує бібліотека QISKit.

2.3 Огляд проблем та алгоритмів для квантових обчислень.

Алгоритм Дойча-Йозса

Алгоритм Дойча-Йозса, вперше введений для демострації можливостей квантового моделювання, також був першим прикладом

квантового алгоритму, який працює краще, ніж найкращий класичний алгоритм. Це показало, що використання квантового комп'ютера як обчислювального інструменту для вирішення конкретної проблеми може мати переваги.

Нам дано приховану булеву функцію f, яка приймає як вхідний рядок бітів і повертає будь-який 0 або 1, це:[7]

$f(\{x0,x1,x2,...\}) \rightarrow 0$ or 1 , де xn це 0 or 1

Властивість даної булевої функції полягає в тому, що вона гарантовано буде збалансованою або постійною. Константна функція повертає всі 0 або всі 1 для будь-якого входу, тоді як збалансована функція повертає 0 рівно для половини всіх вхідних даних і 1 для іншої половини. Наше завдання — визначити, чи є дана функція збалансованою чи постійною.

Зауважте, що проблема Deutsch-Jozsa є n-бітовим розширенням однобітової проблеми Deutsch.

Класичне рішення

Класично, в кращому випадку, два запити до оракула можуть визначити, чи є прихована булева функція, f(x), збалансована: наприклад, якщо ми отримуємо обидва $f(0,0,0,...) \rightarrow 0$ and $f(0,0,0,...) \rightarrow 0$ and $f(1,0,0,...) \rightarrow 1$ то ми знаємо, що функція збалансована, оскільки ми отримали два різних вихідні дані f(x) є збалансованим: наприклад, якщо ми отримуємо обидва $f(0,0,0,...) \rightarrow 0 f(0,0,0,...) \rightarrow 0$ if $(1,0,0,...) \rightarrow 1$ $f(1,0,0,...) \rightarrow 1$, то ми знаємо, що функція збалансована, оскільки ми отримали два різних виходи.

У гіршому випадку, якщо ми продовжимо бачити той самий вихід для кожного входу, який ми намагаємося, нам доведеться перевірити рівно половину всіх можливих вхідних даних плюс один, щоб бути впевненим, що

f(x) є постійним. Оскільки загальна кількість можливих вхідних даних становить 2^n, це означає, що нам потрібно 2^{n-1}+1 пробних вхідних даних, щоб переконатися, що f(x) є постійним у гіршому випадку. Наприклад, для 4-бітового рядка, якщо ми перевірили 8 із 16 можливих комбінацій, отримавши всі 0, все ще можливо, що вхід 9 повертає 1 і f(x) є збалансованим. Імовірно, це дуже малоймовірна подія. Насправді, якщо ми отримуємо той самий результат постійно поспіль, ми можемо виразити ймовірність того, що функція є постійною як функція вхідних даних k, як: f(x) є постійним. Оскільки загальна кількість можливих входів дорівнює 2n, це означає, що нам потрібно 2n−1+1 пробні дані, щоб переконатися в цьому f(x) у гіршому випадку є постійною. Наприклад, для 4-бітовий рядок, якщо ми перевірили 8 із 16 можливих комбінації, отримання всіх 0, все ще можливо, що 9th введення повертає a 1 if(x) є збалансованим. Імовірно, це дуже малоймовірна подія. Насправді, якщо ми отримуємо один і той самий результат постійно поспіль, ми можемо виразити ймовірність того, що функція є постійною як функція від k входи як:

$$P_{\text{constant}}(k) = 1 - \frac{1}{2^{k-1}} \qquad \text{for } 1 < k \leq 2^{n-1}$$

Квантове рішення

Використовуючи квантовий комп’ютер, ми можемо вирішити цю проблему зі 100% впевненістю лише після одного виклику функції f(x), за умови, що у нас є функція f, реалізована як квантовий оракул, який відображає стан $|x\rangle|y\rangle \text{ to } |x\rangle|y \oplus f(x)\rangle,$ де $\oplus$ - додавання за модулем 2. Нижче наведено загальну схему для алгоритму Дойча-Йозса.[8]

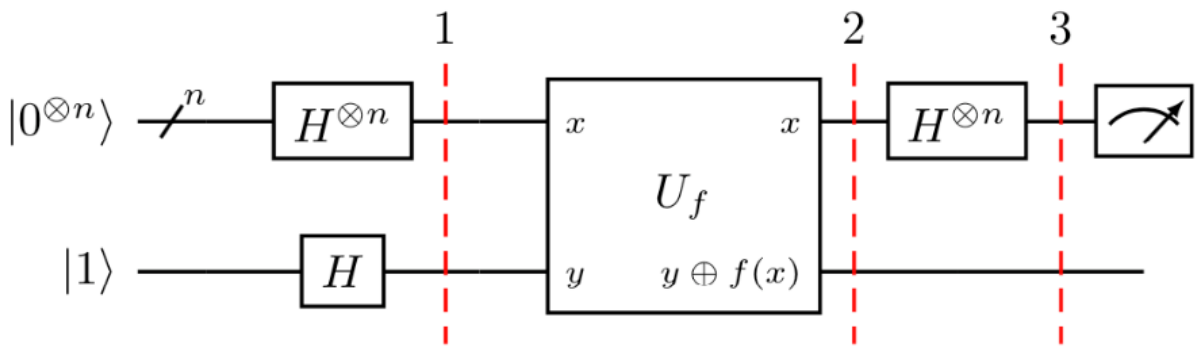


Рисунок 2.9– Демонстрація квантового рішення алгоритму

Тепер давайте розглянемо кроки алгоритму

1. Підготуйте два квантові регістри. Перший – це n-кубітний

$$|\psi_0\rangle = |0\rangle^{\otimes n}|1\rangle$$

регістр ініціалізовано до |0⟩ , а другий — це однокубітний регістр, ініціалізований до |1⟩ формула (1) :

(1)

2. Застосуйте вентиль Адамара до кожного кубіта формула (2):

$$|\psi_1\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle \left(|0\rangle - |1\rangle\right)$$

(2)

3. Застосуйте квантовий с $|x\rangle|y\rangle$ to $|x\rangle|y \oplus f(x)\rangle$:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle(|f(x)\rangle - |1 \oplus f(x)\rangle)$$
$$= \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} (-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle)$$

4. У цей момент другий однокубітний регістр можна ігнорувати. Застосуйте вентиль Адамара до кожного кубіту в першому регістрі формула(4):

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \left[\sum_{y=0}^{2^n-1} (-1)^{x \cdot y} |y\rangle \right]$$
$$= \frac{1}{2^n} \sum_{y=0}^{2^n-1} \left[\sum_{x=0}^{2^n-1} (-1)^{f(x)} (-1)^{x \cdot y} \right] |y\rangle \quad (4)$$

5. Виміряйте перший регістр. Зверніть увагу, що ймовірність вимірювання

який оцінює до 1 якщо f(x) є постійним і 0 якщо f(x) є збалансованим.[9]

Алгоритм Бернштейна-Вазірані

Алгоритм Бернштейна-Вазірані, можна розглядати як розширення алгоритму Дойча-Йозса, який ми розглянули в останньому розділі. Це показало, що використання квантового комп'ютера як обчислювального інструменту для більш складних задач може мати рішення.[10]

Нам знову надається функція f чорного ящика (x), який приймає як вхідний рядок бітів і повертається 0 або 1 це формула (1):

$$f(\{x_0, x_1, x_2, \dots\}) \to 0 \text{ or } 1 \text{ where } x_n \text{ is } 0 \text{ or } 1$$

(1)

Замість того, щоб функція була збалансованою або постійною, як у задачі Дойча-Йозса, тепер функція гарантовано повертає порозрядний добуток введення x, $f(x) = s \cdot x \pmod 2$

Іншими словами, за умови введення, яке ми маємо знайти

Як класична реверсивна схема оракул Бернштейна-Вазірані виглядає так:

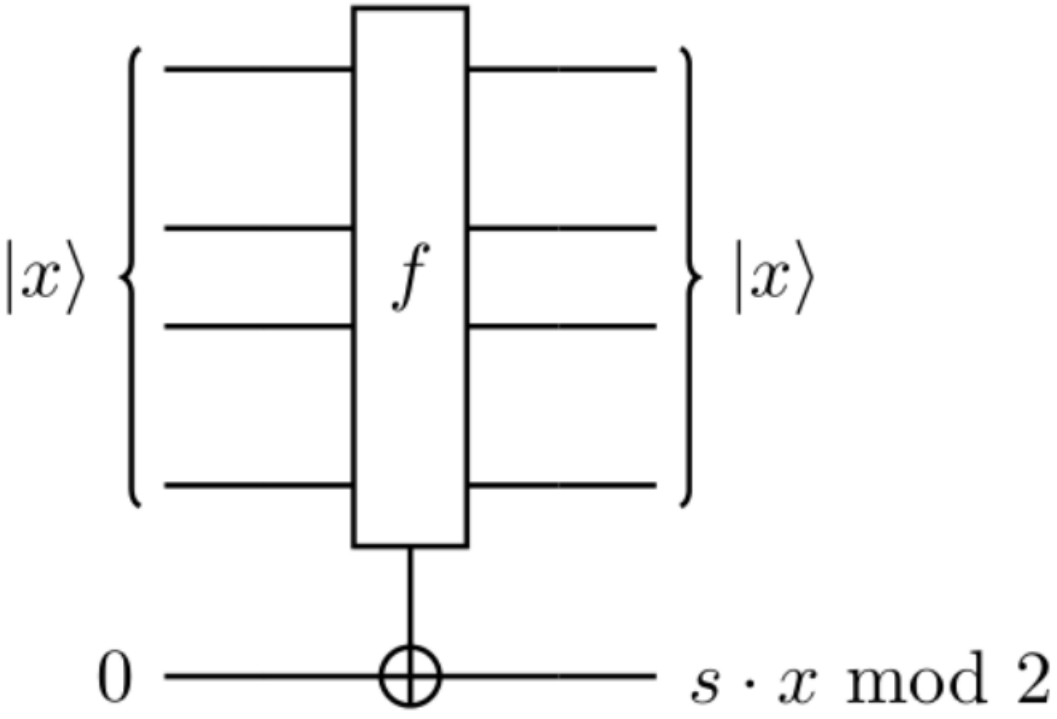


Рисунок 2.10 – Схема роботи алгоритму 1 [33]

Класичне рішення

Класично оракул повертає :

$$f_s(x) = s \cdot x \mod 2$$

Даємо на вхід x. Таким чином, прихований бітовий рядок s можна виявити шляхом запиту оракула з послідовністю вводів:

Input(x)
100...0
010...0
001...0
000...1

Де кожен запит виявляє різну частину s. Наприклад, при x = 1000...0 можна отримати молодший біт s, при x = 0100...0 можна знайти наступний найменш значущий і так далі. Це означає, що нам потрібно буде викликати функцію f(x), n разів.

Квантове рішення

Використовуючи квантовий комп'ютер, ми можемо вирішити цю проблему зі 100% впевненістю лише після одного виклику функції f(x).[11]

Квантовий алгоритм Бернштейна-Вазірані для пошуку прихованого бітового рядка дуже простий:

1. Ініціалізуйте вхідні кубіти
2. Застосуйте вентилі Адамара до вхідного регістра
3. Запитайте пророцтво
4. Застосуйте вентилі Адамара до вхідного регістра
5. Виміряйте

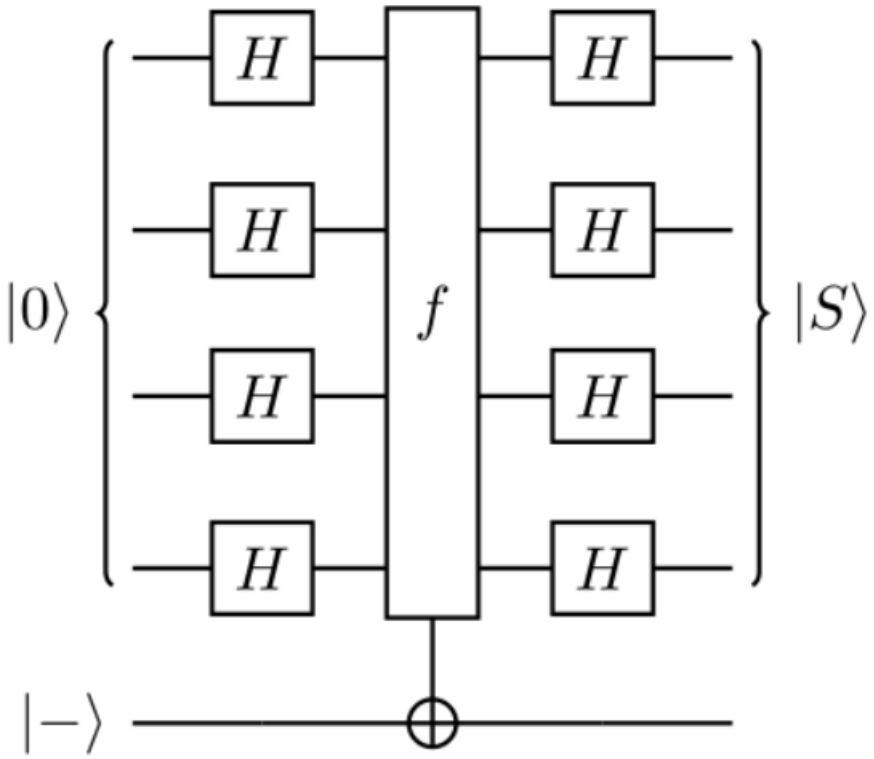


Рисунок 2.11 – Схема роботи алгоритму 2 [33]

Це основна схема для вирішення такого вида проблеми. І як ми можемо бачити на симульованій квантовій логіці він вирішується легше і швидше.

2.4 Детальний розгляд бібліотеки Qiskit

Зараз у сфері програмного забезпечення виділяються явні фаворити порівняно з іншими пропозиціями мов програмування та бібліотек для квантових систем. Через те, що великі компанії вкладають більше ресурсів в розробку та підтримку програмного забезпечення та проводять величезні івенти для знайомства з їх продуктом ми набагато швидше звикаємо до цих продуктів. Виходить так, що в основному не дивлячись на різні переваги малих проектів, саме зараз серйозно можна зрівнювати дві технології і це IBM Qiskit та Microsoft QDK.[14]

Тож оцінимо ці продукти за наступних параметрах:

Підхід

Qiskit починає з квантових процесорів і квантових симуляторів і створює значну бібліотеку алгоритмів для цілей вивчення квантової фізики та використання квантових обчислень. QDK, з іншого боку, починається з Visual Studio і Microsoft Office і дозволяє вам розробляти все, що ви можете придумати, використовуючи QPU для спеціалізованих обчислень, схожих на те, як ви можете використовувати GPU і TPU для інших спеціалізованих обчислень.

Іншими словами, Qiskit є «квантовим» у своїй основі, тоді як QDK насправді є подальшим розширенням значної платформи розробки.

Абстракція

Qiskit дозволяє розкладати числа на множники за допомогою одного рядка коду. Можливо, більш вражаючим, є те, що ви можете створити оракул Гровера з однорядковим виразом. Відверто кажучи, я не знаю, чи QDK дозволяє реалізувати подібні можливості, але в рузультаті слідкування за тенденціями Qiskit, безумовно, виходить, що від пояснення того, як працюють алгоритми, до абстрагування від того, як вони працюють можна прийти дуже легко і швидко.

Документація QDK має на увазі подібну мету: потрібно більше писати алгоритми, а не будувати схеми і розглядати їх як візуалні приклади. Однак, документація та відео показують, як писати схеми, тому не відомо, як компанія внутрішньо узгоджує цю невідповідність.

Кубіти

Qiskit, очевидно, віддає перевагу надпровідним кубітам IBM, в той час як постачальники Azure Quantum мають системи іонів. Якщо метою дослідження чи отримання досвіду роботи є використання квантових додатків, вам дійсно не потрібно турбуватися про це. Навіть квантові мови асемблера абстраговані набагато вище цього. Єдина програма, для якої можна сказати є багато актульності в цьому питанні - це Qiskit Pulse, тому якщо вам потрібна функціональність для лабораторного використання, то Qiskit, здається, стане для вас вибором.[15]

Додатки

Швидке порівняння облікових записів GitHub показує набагато більшу широту для Qiskit, ніж для QDK. Тож, незалежно від того, чи зацікавлені дослідники в дослідженні всіх областей чи спеціалізуватися лише на одній, Qiskit має більше коду для вас. На додачу до цього, Qiskit має великий підручник з підручниками та інтерактивними підручниками, а

також навчальні відео на YouTube. В результаті аналізу деяких підручників QDK, виявилось, що цільова аудиторія не дуже зрозуміла; зміст був вступним, але пояснення були поверховими, наче людина має вже бути знайома з змістом та знати всі необхідно основи для роботи з книгою.

Але, це можна сказати лише про додатики. Якщо перевірити Qiskit на GitHub, мождна побачите, всередині можна знайти достатньо багато користої інформіціх. Крім того, всередині порталу IBM Quantum величезна кількість покинутих інструментів для симуляції кватнових систем, але звичайний новачок в дослідженні не зможе з цим розібратися, бо інтерфейс не інтітивний, а симі інструменти зроблені вже скореше за всюго для людей з досвідом і людині потрібно спочатку знаходити джерела по навчанню саме на цій платформі, а потім практикувати.

Спільнота

На мою думку, загальновизнано, що Qiskit має найбільшу спільноту розробників і ми можемо побачити це по мета-аналізу соціальний мереж та кількості написання статей по темі. З його облікового запису GitHub видно, що він також має найбільшу спільноту учасників. Ми можемо легко прослідкувати лише за одним проектом, і рівень активності може насправді здивувати, бо люди працюють над задачами невпино. Через це, можна в цілому очікувати, що переваги Qiskit в абстракції та додатках прискоряться, оскільки його спільнота стає дедалі більше.

Ясність

Здається, що спільнота Qiskit складається переважно з фізиків, математиків та інших учених. Вони спілкуються між собою та пишуть стітті з математикою, лінійною алгеброю та такими термінами, як гамільтоніан, ермітовий і комплексно спряжений. Microsoft приділяє принаймні деякий час коду самому по собі, бо це є одною з найважливіших частин розуміння

теми для людей з бекграундом в програмуванні, багато хто з них не знаю фізику і вони не можуть стати частиною активного контакту з іншими дослідниками в області. Тому ваш досвід може бути важливим фактором у визначенні того, яку структуру легше прийняти.

Інструменти

Qiskit використовує блокноти Jupyter і мову Python. QDK, з іншого боку, використовує ноутбуки Jupyter, Visual Studio, Microsoft Office, тож мож сміливо сказти, що це наспрвді багато. Я використовую продукти Microsoft досить довго, тому я був би вражений, звичайна людина не змогла знайти доступ до таких інструмнтів. Більш того, QDK дозволяє розробляти програми, які можна публікувати в хмарі як веб-сервіс. Не можна сказати, що розробник не можете зробити це за допомогою Qiskit, але додавання таких класичних функцій починає переміщувати переваги на Microsoft. [31]

Мови

Qiskit дозволяє використовувати лише Python. QDK дозволяє використовувати лише Q#, але навколо Q# у вас є вибір: C#, J#, Visual Basic, а також Python. Тому, що стосується класичних функцій ваших програм, QDK може робити все, що може Qiskit, і набагато більше.

Вартість

Qiskit можна використовувати безкоштовно.

Azure Quantum, з іншого боку, згадує безкоштовну пробну версію. І якщо Microsoft збирається надати «до 10 000 доларів США кредитів для використання на квантовому обладнанні», їх цільовою аудиторією повинні бути установи, а не окремі особи чи маленькі дослідницькі групи студентів.

Отже, якщо людина новачок у квантових обчисленнях, потрібно зважити всі данні, але більш зручним та доступним для звичайного дослідника є використання безкоштовного використання Qiskit. Пізніше можна вибрати між партнерською програмою IBM і Azure Quantum.

Тим не менш всі переваги Qiskit, крім вартості, також належать QDK. У цьому випадку людина також можете використовувати Cirq з QDK. Таким чином, QDK надає вам квантові програми як Qiskit, так і Cirq, а також класичні програми Visual Studio, Microsoft Office тощо.

Можна сказати також, що одна з наайкращих функцій Cirq, це можливість імпорту з Quirk. Це чудовий інструмент для швидкого створення та тестування малих схем. Але замість того, щоб створювати і тестувати за допомогою Quirk, а потім повторно створювати за допомогою Qiskit, можна імпортувати свою схему в Cirq, а потім імпортувати Cirq в QDK.

Висновок

Qiskit — це безкоштовна система квантових обчислень для фізиків, хіміків і математиків. Якщо ви хочете зосередитися на квантовому аспекті дослідження це, мабуть, найкращий вибір. Qiskit з усіма фреймворками контролю якості. Qiskit має найбільшу спільноту розробників, а його GitHub величезний, тому це, можливо є найкращім вибором для початківця, чи людини, яка цікавиться темою.

QDK, з іншого боку, є частиною загальної обчислювальної системи для комп’ютерників. Можна розробити все, що тільки можете собі уявити, а також використовувати квантові обчислення. І хоча можна використовувати Qiskit через QDK, це безкоштовно лише протягом пробного періоду.

Але тим не менш можна було б рекомендувати його для початку розробки Qiskit буде правильним рішенням саме через наявність досвіду з мовою програмування Python.

РОЗДІЛ 3 ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

Алгоритм Шора відомий розкладанням цілих чисел за поліноміальний час. Оскільки найвідоміший класичний алгоритм вимагає суперполіноміального часу для розкладання добутку двох простих чисел, широко використовувана криптосистема, RSA, покладається на те, що розкладання на множники неможливо для достатньо великих цілих чисел.

У цьому розділі ми зосередимося на квантовій частині алгоритму Шора, яка фактично вирішує проблему знаходження періоду. Оскільки задачу розкладання можна перетворити на задачу знаходження періоду за поліноміальний час, ефективний алгоритм знаходження періоду також можна використовувати для ефективного розкладання цілих чисел на множники. Наразі цього достатньо, щоб показати, що якщо ми можемо обчислити період ax mod N ефективно, то ми також можемо ефективно фактор. Оскільки пошук періоду сам по собі є гідною проблемою, ми спочатку вирішимо її, а потім обговоримо, як це можна використати для врахування факторів у розділі 5. [32]

Для початку імпортуємо всі необхідні бібліотеки для розробки програми:

```
import matplotlib.pyplot as plt

import numpy as np

from qiskit import QuantumCircuit, Aer, transpile, assemble

from qiskit.visualization import plot_histogram

from math import gcd

from numpy.random import randint

import pandas as pd

from fractions import Fraction

print("All elements have been imported")
```

3.1 Задача: Розглянути проблему

Поглянемо як виглядає періодична функція -

f(x) = a^x mod N

Де a та N – позитивні числа, а менше ніж N, та в них немає схожих факторів. Період, або порядок (r) це найменше (ненульове) ціле число таке як:

a^r mod N = 1

Ми можемо визначити, що приклад цієї функції, зображений на графіку нижче показує період функції так як він залежить від наведених

парметрів. Зверніть увагу, що лінії між точками дають можливість побачити періодичність і не представляють проміжні значення між маркерами x.

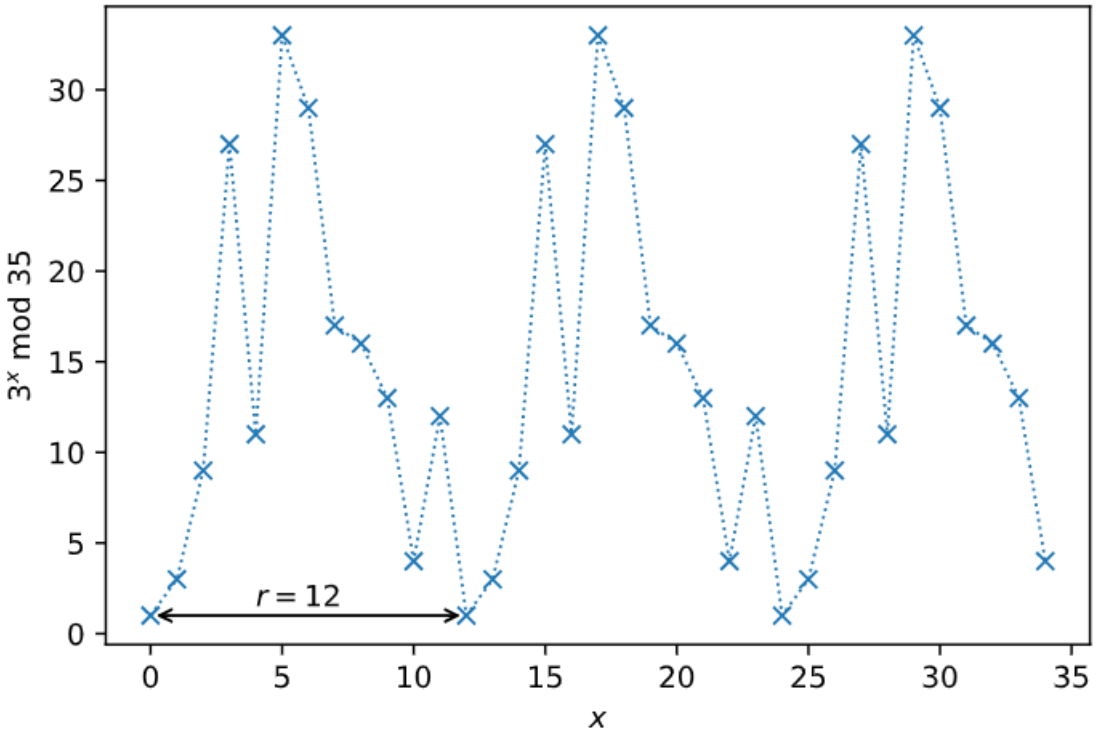


Рисунок 3.1 – Приклад періодичної функції в алгоритмі Шора [29]

3.2 Розглянемо матиматичне рішення проблеми

Тож щоб підійти до вирішення Шора нам потрібно використати квантову фазу оцінки на унітарному операторі:

$U\,|y\rangle \equiv |a\ y \bmod N\rangle$

Щоб зрозуміти, наскільки це важливо, потрібно трохи розібратися, як може виглядати власний стан U. Допустимо, ми починаємо в стані |1⟩ , тож можна побачити, що кожне наступне застосування U буде множити стан нашого реєстру на a (mod N), а після r додатку ми прийдемо до стану |1⟩ ще раз. Нариклад, a = 3, N = 35:

$$\begin{aligned} U|1\rangle &= |3\rangle \\ U^2|1\rangle &= |9\rangle \\ U^3|1\rangle &= |27\rangle \\ &\vdots \\ U^{(r-1)}|1\rangle &= |12\rangle \\ U^r|1\rangle &= |1\rangle \end{aligned}$$

Рисунок 3.2 – Ефект успіху додатків з U

Тож суперпозиція стану в циклі (|u0⟩) буде своїм станом в U:

$$|u_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |a^k \bmod N\rangle$$

Такий власний стан може мати власне значення в виді 1, що може здатися тривіальним. Трохи цікавішим власним станом можна вважати

$$|u_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i k}{r}} |a^k \bmod N\rangle$$

$$U|u_1\rangle = e^{\frac{2\pi i}{r}} |u_1\rangle$$

такий що, у фазі різна для кожного з цих обчислювальних базових станів. Отож, можемо розглянути випадок, у якому фаза Kth та стан пропорційний K (Рисунок 3.2):

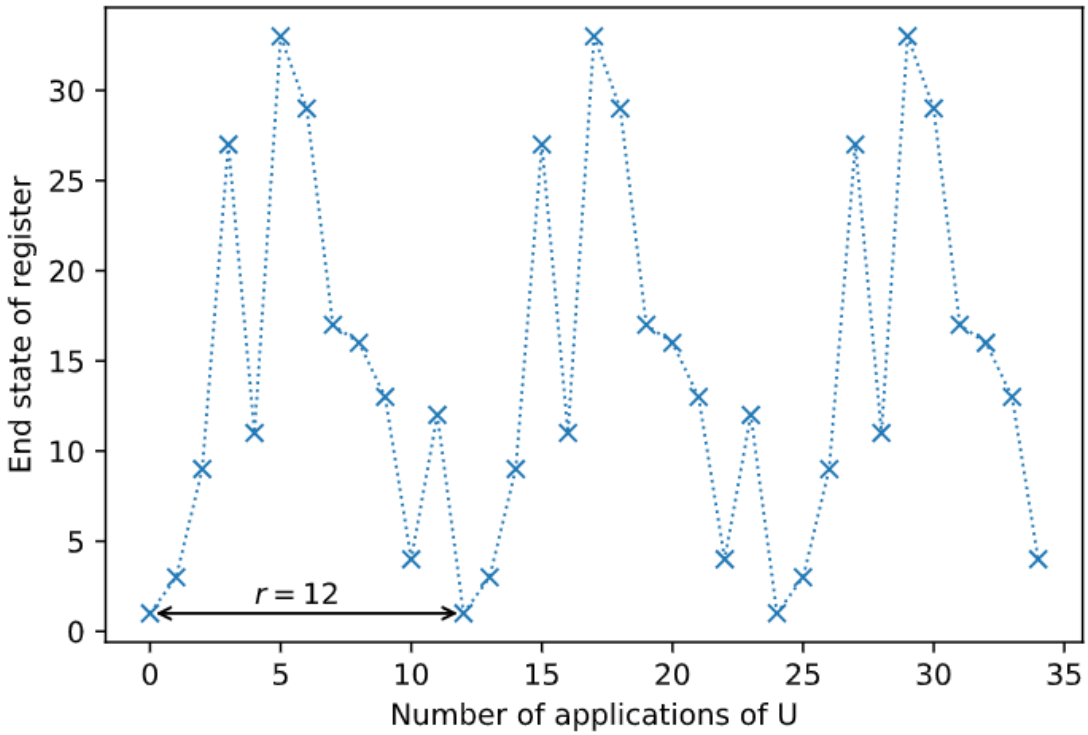


Рисунок 3.2 – Приклад періодичної функції в алгоритмі Шора [29]

Саме таке власне значення має для нам більший інтерес, оскільки воно містить r. Таким чином, r має бути включено, щоб бути впевненим, що різниця фаз між r обчислювальні базисні стани рівні. Це не єдиний власний стан з такою поведінкою; щоб узагальнити це далі, ми можемо помножити

ціле число s, до цієї різниці фаз, яка відобразиться в нашому власному значенні:

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} |a^k \bmod N\rangle$$

$$U|u_s\rangle = e^{\frac{2\pi i s}{r}} |u_s\rangle$$

Тепер ми маємо унікальний стан для кожного власного цілого числа r, де

$$0 \leq s \leq r-1$$

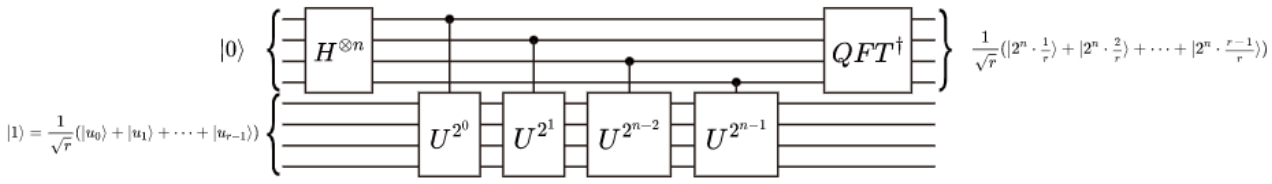
Далі виходить правильно, якщо зсумувати всі власні стани, різні фази скасовують усі обчислювальні базові стани, крім |1⟩:

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle$$

Затим, що обчислювальний базис стану |1⟩ це суперпозиція цих станів, що означає якщо ти робиш QPE на U, використовуючи стан |1⟩ ми заміримо фазу:

$$\phi = \frac{s}{r}$$

Де s це випадкове ціле число від 1 і r – 1. Нарешті ми використовуємо алгоритм безперервних дробів ϕ щоб знайти r. Принципова схема виглядає



так (майте на увазі, що ця діаграма використовує конвенцію Qiskit про порядок упорядкування кубітів):

Тепер вже ми зможемо проедмострувати алгоритм Шора за допомогою симуляторів Qiskit. Щоб розробити цю программу ми маємо надати демонстрації схеми для U, але в кінці розділу буде проілюстровано, як схеми для U^{2^j} можна швидко та зрозуміло побудувати.

3.3 Реалізація Qiskit

В цій ситуації ми розв'язуємо задачу на знаходження періоду для a = 7 та N = 15. Ми ппредставимо схему для U де:

$$U|y\rangle = |ay \bmod 15\rangle$$

Щоб задати U^x ми маємо повторимо коло x разів. У другій частині практичного завдання ми обговоримо загальний метод ефективного створення цих схем. Функція c_amod15 повертає керований елемент U протягом a, повторюваного часу потужності.[33]

Наступна частина коду:

```
def c_amod15(a, power):
    """Контрольоване множення за допомогою mod 15"""
    if a not in [2,4,7,8,11,13]:
        raise ValueError("'a' must be 2,4,7,8,11 or 13")
    U = QuantumCircuit(4)
    for iteration in range(power):
        if a in [2,13]:
            U.swap(0,1)
            U.swap(1,2)
            U.swap(2,3)
        if a in [7,8]:
            U.swap(2,3)
```

В цей раз ми використаємо 8 обчислювальних кубітів і отримаємо наступні рзультати:

```
    c_U = U.control()
    return c_U
```

```
        if a in [7,8]:
            U.swap(2,3)
            U.swap(1,2)
            U.swap(0,1)
        if a in [4, 11]:
            U.swap(1,3)
            U.swap(0,2)
        if a in [7,11,13]:
            for q in range(4):
                U.x(q)
    U = U.to_gate()
    U.name = "%i^%i mod 15" % (a, power)

# Вказуємо змінні
n_count = 8  # Кількість обчислювальних кубітів
a = 7
```

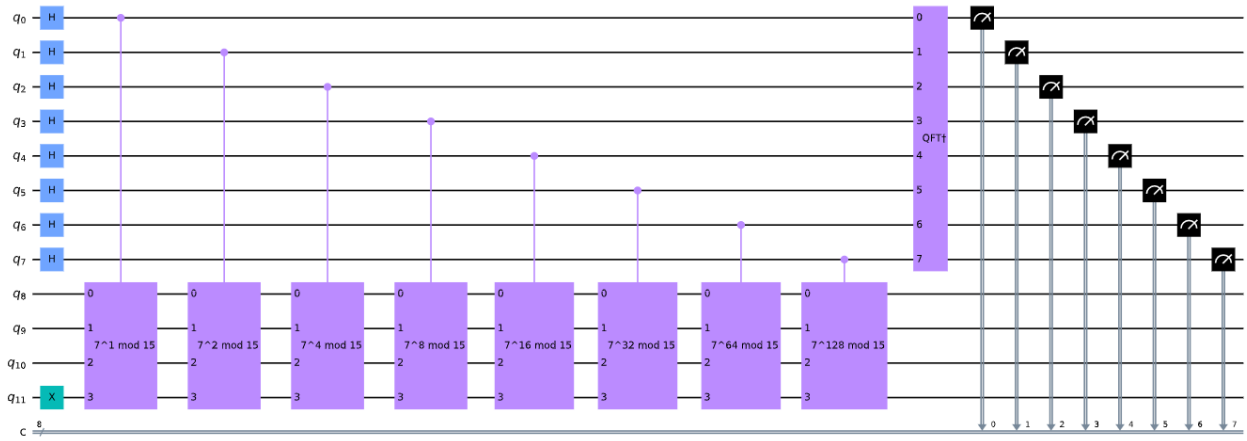
Маємо імпортувати схему для QFT:

```python
def qft_dagger(n):
    """n-кубіт QFTdagger перший n кубітів в схемі"""
    qc = QuantumCircuit(n)
    # Потрібно звернути увагу та не забути свопи
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cp(-np.pi/float(2**(j-m)), m, j)
        qc.h(j)
    qc.name = "QFT†"
    return qc
```

За допомогою цих блоків побудуємо зрозумілий та псолідовний цикл для алгоритму Шора. І після певного редагування та пояснення він має виглядати так:

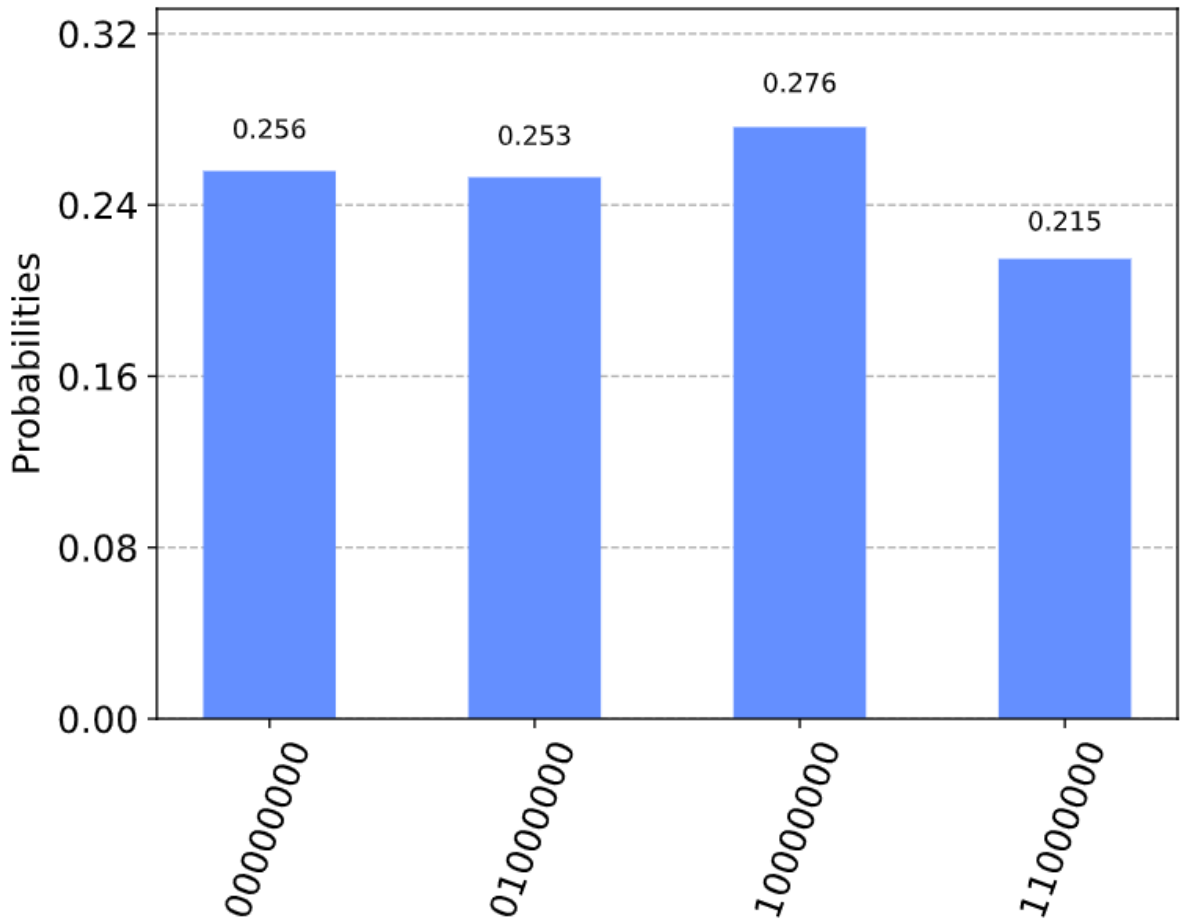
```
# Створим квантовий цикл n_count рахуючи кубіти
# плюс 4 кубіти для U які задіяні
qc = QuantumCircuit(n_count + 4, n_count)
# Ініціюємо рахуючи кубіти
# в стані |+>
for q in range(n_count):
    qc.h(q)
# І допоміжний регістр в стані |1>
qc.x(3+n_count)
# Виповнити контрольовані-U оператори
for q in range(n_count):
    qc.append(c_amod15(a, 2**q),
              [q] + [i+n_count for i in range(4)])
# Виповнити інверсійне-QFT
qc.append(qft_dagger(n_count), range(n_count))
# Цикл виміру
qc.measure(range(n_count), range(n_count))
qc.draw(fold=-1)  # -1 означає не складається
```

Давайте подивимось на результати вимірювань в графічному вигляді:



І текстовий вивід:

```
aer_sim = Aer.get_backend('aer_simulator')

t_qc = transpile(qc, aer_sim)

qobj = assemble(t_qc)

results = aer_sim.run(qobj).result()

counts = results.get_counts()

plot_histogram(counts)
```



Так як в нас є 8 кубітів, ці результати відповідають виміряним фазам зображеним в данному коді:

```
rows, measured_phases = [], []

for output in counts:

    decimal = int(output, 2) # Конвертувати (base 2) строку в десятині

    phase = decimal/(2**n_count)  # Знайти відповідні власні значення

    measured_phases.append(phase)

    # Додати ті значення в строку в таблиці:

    rows.append([f"{output}(bin) = {decimal:>3}(dec)",

                 f"{decimal}/{2**n_count} = {phase:.2f}"])

# Вивести рядки в таблиці

headers=["Register Output", "Phase"]

df = pd.DataFrame(rows, columns=headers)

print(df)
```

Результат обчислення:

```
           Register Output           Phase
0  00000000(bin) =   0(dec)    0/256 = 0.00
1  01000000(bin) =  64(dec)   64/256 = 0.25
2  11000000(bin) = 192(dec)  192/256 = 0.75
3  10000000(bin) = 128(dec)  128/256 = 0.50
```

Після попередніх дій, ми можемо імплеменувати алгоритм безперервних дробів, щоб спробувати знайти s і r. Також ми можему

використати функціональність, що вбудована в Python: використовуємо модуль fractions, щоб перетворити float в об’єкт Fraction, наприклад:

```
Fraction(0.666)
```

Має результат:

```
Fraction(5998794703657501, 9007199254740992)
```

Так як наша програма видає нам дроби, що повертають результат точно (у цьому випадку 0,6660000...), це може дати грубі та не потрібно великі результати, як показано на зображенні вище. Але, ми можемо використати такий метод як .limit_denominator(), щоб мати дріб, який точно нагадує нам стандартний float, зі знаменником нижче певного значення:

Після такого застосування вивід має виглядати краще. Порядок (r) має бути меншим за N, тому ми встановимо максимальний знаменник рівним

```
# Отримати долю, яка більше всього нагадує 0,666
# зі знаменником < 15
Fraction(0.666).limit_denominator(15)
```

```
Fraction(2, 3)
```

15:

```
rows = []
for phase in measured_phases:
    frac = Fraction(phase).limit_denominator(15)
    rows.append([phase, f"{frac.numerator}/{frac.denominator}",
frac.denominator])
# Вивести як таблицю
headers=["Phase", "Fraction", "Guess for r"]
df = pd.DataFrame(rows, columns=headers)
print(df)
```

	Phase	Fraction	Guess for r
0	0.00	0/1	1
1	0.25	1/4	4
2	0.75	3/4	4
3	0.50	1/2	2

Можна побачити, що два з виміряних власних значень дали нам правильний результат: r=4, і очевидно, що алгоритм Шора має невелику погрішніть та вірогідність отримати помилку. Ці неточні та помилкові результати пов’язані з тим, що s = 0, або тому, що s і r не є взаємно простими і замість r нам дається коефіцієнт r. В результаті тестування програми та вивченя алгоримтму, можна знайти наступне простіше рішення — просто повторити експеримент, поки не отримаємо задовільний результат для r. Виходить так, що квантова система має можливать помилитися, але її швитдкість нсастільки велика, що ми можемо запускати програму величезну кількість разів, поки вона не отримає ідеальні умови і не видасть правильну відповідь.

Перейдемо до наступного етапу.

1. Модульне підведення до степеня

Можна звернути увагу на те, що метод створення U^{2^j} вхід за рахунок повторення U зростає експоненціально з j і не призведе алгоритм поліноміального часу. Тож, щоб вирішити це питання, нам потрібно зробити поліноміальний оператор:

$$U^{2^j}|y\rangle = |a^{2^j} y \bmod N\rangle$$

що росте поліноміально з j. На щастя, розраховуючи: $a^{2^j} \bmod N$ ефективність можлива. Звичайні ЕОМ можуть використовувати алгоритм, який ми можемо знати під назвою повторне зведення в квадрат для обчислення експоненти. В цьому конкретному випадку, маємо справу лише з експонентами виду 2^j, алгоритм повторення квадратури стає дуже простим, ось його реалізація в програмі:

```
def a2jmodN(a, j, N):
    """Рахуємо a^{2^j} (mod N) повторюючи квадратуру"""
    for i in range(j):
        a = np.mod(a**2, N)
    return a
```

Вводимо данні:

```
a2jmodN(7, 2049, 53)
```

Отримуємо результат:

```
47
```

При умові, що в Python наявний ефективний алгоритм, тож можна використовувати той самий алгоритм на квантовому комп'ютері. Але прикро те, що незважаючи на поліноміальне масштабування за допомогою модульні кола піднесення до ступеня не є достатньо простими і можна назвати це слабкою віхою в алгоритмі Шора.

5. Факторинг із визначення періоду

На практиці далеко не всі проблеми факторингу можна назвати складними, тож ми можемо миттєво помітити парне число і знати, що одним з його множників є 2. Таким чином це майже не зайняло в нас часу, щоб викинути це число з обчислювального процесую. В рельності існують певні критерії для вибору чисел, які важко розкласти на множники, але головна думака полягає в тому, щоб вибрати саме добуток двох великих простих чисел.

Спершу потрібно перевірити загальний алгоритм розкладання, чи предмет того, чи існує ярлик для розкладання цілого числа на множники (Чи число парне? Чи є число у формі N = a^b?), перш ніж використовувати алгоритм Шора для найскладнішого сценарію. Базуючись на цих викладках ми маємо приділити більше уваги квантовій частині алгоритму, таким чином перейдемо безпосередньо до випадку, коли N є добутком двох простих чисел.

Приклад: Факторинг 15

Щоб зрозуміти та продемонструвати приклад розкладання на невелику кількість кубітів, ми розкладемо 15, яке, як всім відомо, є добутком не дуже великих простих чисел 3 і 5.

Першим кроком є вибір випадкового числа, a, між 1 і N – 1[28] :

```
np.random.seed(1) # Дізнаємося чи відновлюється результат

a = randint(2, 15)

print(a)
```

Отримуємо очікуваний результат результат – 7

Далі ми маємо перевірити, чи не є це вже нетривіальним фактором N.

Далі ми виконуємо алгоритм пошуку порядку Шора для a = 7 і N = 15.

Такж ми маємо мати на увазі той факт, що фаза, яку ми вимірюємо, буде s/r

$$a^r \bmod N = 1$$

Маємо такий код програми:

```
    aer_sim = Aer.get_backend('aer_simulator')

    # Налаштування memory=True нижче дозволяє нам побачити список
кожного послідовного зчитування
    t_qc = transpile(qc, aer_sim)

    qobj = assemble(t_qc, shots=1)

    result = aer_sim.run(qobj, memory=True).result()

    readings = result.get_memory()

    print("Register Reading: " + readings[0])

    phase = int(readings[0],2)/(2**n_count)

    print("Corresponding Phase: %f" % phase)
```

```
    qc.append(qft_dagger(n_count), range(n_count)) # inverse-QFT

    qc.measure(range(n_count), range(n_count))

    # Симулюємо результати
```

На цьому етапі ми можемо легко знайти припущення для r:

```
phase = qpe_amod15(a) # Фаза = s/r

Fraction(phase).limit_denominator(15) # Демонстрація має видати нам
результат по r
```

Результат роботи програми:

```
Register Reading: 01000000
Corresponding Phase: 0.250000

Fraction(1, 4)
```

```
frac = Fraction(phase).limit_denominator(15)

s, r = frac.numerator, frac.denominator

print(r)
```

Тепер у нас є r, ми можемо використовувати це, щоб знайти коефіцієнт N. Оскільки [28]: $a^r \bmod N = 1$

то $(a^r - 1) \bmod N = 0$

А це означає N треба розділити a^r-1. А якщо r також парне, то можна записати: $a^r - 1 = (a^{r/2} - 1)(a^{r/2} + 1)$

```
guesses = [gcd(a**(r//2)-1, N), gcd(a**(r//2)+1, N)]

print(guesses)
```

Клітинка нижче повторює алгоритм, поки не буде знайдено хоча б один коефіцієнт 15.

А ось і сама програма по знаходженню факторіалів:

```
a = 7

factor_found = False

attempt = 0

while not factor_found:

    attempt += 1

    print("\nAttempt %i:" % attempt)

    phase = qpe_amod15(a) # Фаза = s/r

    frac = Fraction(phase).limit_denominator(N) # демонструємо r

    r = frac.denominator

    print("Result: r = %i" % r)

    if phase != 0:

        # Вгадування факторіалів gcd(x^{r/2} ±1 , 15)

        guesses = [gcd(a**(r//2)-1, N), gcd(a**(r//2)+1, N)]

        print("Guessed Factors: %i and %i" % (guesses[0], guesses[1]))

        for guess in guesses:

            if guess not in [1,N] and (N % guess) == 0:

              # Перевірте, чи є припущення факторіалом

                print("*** Non-trivial factor found: %i ***" % guess)

                factor_found = True
```

Результат роботи програми:

```
Attempt 1:
Register Reading: 00000000
Corresponding Phase: 0.000000
Result: r = 1

Attempt 2:
Register Reading: 11000000
Corresponding Phase: 0.750000
Result: r = 4
Guessed Factors: 3 and 5
*** Non-trivial factor found: 3 ***
*** Non-trivial factor found: 5 ***
```

Висновок до розділу 3

В результаті проведеної роботи щодо розгляду проблематики та можливих варіантів розвзязування задачі ми розібрались з квантовою частиною алгоритма Шора, яка насправді змогала вирішити проблему знаходження періоду та розкладання на множники. Оскільки задачу розкладання можна перетворити на задачу знаходження періоду за поліноміальний час, ми змогли знайти та впровадити такий алгортм. Також ми розібрались з математичним аспектом алгоритму та скористались програмою для ефективного розкладання цілих чисел на множники. Таким чином, я вважаю, що цього має бути достатньо, щоб продемонструвати перспективність та логіку мислення в такому виді алгоритмів, також те що ми можемо ефективно занйти факторіал. Так як знаходження періоду саме по собі є складною задачею, ми спочатку вирішили її, а потім обгрунтували, як це можна застосувати для вирухування факторіалів.

ВИСНОВОК

У першому розділі даної роботи було розглянуно предметну область й був зроблений екскурс в теоретичну частину питання, що стосувалася даної теми роботи. Було продемонстровано підходи, щодо строрення квантових комп'ютерів. Було доодатково розглянуто моделі та алгоритми квантового обчислення. Важливим етапоп для розіміння актулаьності та перспективності цієї гілки комп'ютеорної науки є розгляд практитчних та конкретних механізмів роботи обчислень на фізичному рівні.

У другому розділі даної роботи було розглянуто методи й інструменти для реалізації існуючої пробелматики. Особливо детально описані різні платформи симуляціїї квантових обчислень, зокрема QX симулятор, та Qiskit бібліотека для мови програмування Python. Більш того, ми зробили огляд багатьох вже існуючих та ефективних інструмнтів для симуляції квантових обчислеть, та зробили їх конкретне прорівняння з оцінкої плюсів та мінусів.

У третьому розділі дипломного проекуту ми алгоритмічно показали етапи розробки програмного забезпечення та підхід до вирішення існуючої задачі. Наведено опис й перелік програмних компонентів. Були отримані конкретні результати, та провелась оцінка та релевантність данного підходу до рішення сучасних обчислювальних пробелем.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Learn Quantum Computing with Python and IBM Quantum Experience 2020 pp. 20-21.

2. Robert Hundt - Quantum Computing for Programmers 2022 pp.21-25

3. ASCR Report on Quantum Computing for Science, Alán Aspuru-Guzik, Wim van

4. Dam, Edward Farhi, Frank Gaitan 2020 pp. 50-65.

5. Quantum Computing: Progress and Prospects 2018 pp. 95-96

6. Quantum Computing for Everyone (The MIT Press) 2019

7. Dancing with Qubits: How quantum computing works and how it can change the world (Packt Publishing) 2019 pp. 125-129

8. Quantum Computing: An Applied Approach (Springer) 2019

9. Quantum Computing since Democritus (Cambridge University Press) 2013 pp. 265-268

10. Quantum Computing for beginners: A Complete beginner's guide to Explain in Easy Way, History, Features, Developments and Applications of New Quantum Computers that will Revolutionize the World (self-published) 2020 pp. 12-15

11. Quantum Computers: Essential Algorithms and Code (O′Reilly) 2015 pp. 15-19.

12. R. Jagannathan, T. S. Santhanam, and R. Vasudevan, "Finite Dimen-sional Quantum Mechanics of a Particle," Int. J. Theor. Phys., vol. 20, p. 755, 1981.

13. Meurice, "A discretization of p-adic quantum mechanics," Communications in Mathematical Physics, vol. 135, no. 2, pp. 303– 312, Jan 1991.

14. A. Singh and S. M. Carroll, "Modeling Position and Momentum in Finite-Dimensional Hilbert Spaces via Generalized Clifford Algebra 1806.10134," 2018.

15. A. Macridin, P. Spentzouris, J. Amundson, and R. Harnik, “Electron-Phonon Systems on a Universal Quantum Computer,” Phys. Rev. Lett., vol. 121, no. 11, p. 110504, 2018.

16. D. Berenstein, “A toy model for time evolving QFT on a lattice with controllable chaos 1803.02396,” 2018.

17. N. M. Atakishiyev, A. U. Klimyk, and K. B. Wolf, “A discrete quantum model of the harmonic oscillator,” Journal of Physics a Mathematical General, vol. 41, no. 8, p. 085201, Feb. 2008.

18. M. Lorente, “Continuous vs. discrete models for the quantum harmonic oscillator and the hydrogen atom,” Physics Letters A, vol. 285, no. 3, pp. 119 – 126.

19. L. Barker, agatay Candan, T. Hakioglu, M. A. Kutay, and H. M. Ozaktas, “The discrete harmonic oscillator, harper’s equation, and the discrete fractional fourier transform,” Journal of Physics A: Mathematical and General, vol. 33, no. 11, p. 2209, 2000. [Online]. Available: http://stacks.iop.org/0305-4470/33/i=11/a=304

20. M. Aunola, “The discretized harmonic oscillator: Mathieu functions and a new class of generalized hermite polynomials,” Journal of Mathematical Physics, vol. 44, no. 5, pp. 1913–1936, 2003. [Online].

21. W. Heisenberg, “Uber quantentheoretische umdeutung kinematischer und mechanischer beziehungen.” Zeitschrift fur¨ Physik, vol. 33, no. 1, pp. 879–893, Dec 1925. [Online]. Available: https://doi.org/10.1007/BF01328377

22. P. Woit, Quantum Theory, Groups and Representations. Springer, 2017. [Online]. Available: http://inference-review.com/article/woits-way

23. Yu. R. Musin, “A Supersymmetric anharmonic oscillator,” Sov. Phys. J., vol. 33, pp. 401–404, 1990.

24. A. Peruzzo, J. McClean, P. Shadbolt, M. Yung, X. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O’Brien, “A variational eigenvalue solver on a quantum processor,” 2013.

25. N. Klco and M. J. Savage, “Digitization of Scalar Fields for NISQ-Era Quantum Computing 1808.10378,” 2018.

26. A. K. Das, “Supersymmetry and Finite Temperature,” Physica, vol. A158, pp. 1–21, 1989.

27. Quantumcomputing.stackexchange.com/questions/1474/what-programming-languages-are-available-for-quantum-computers

28. Introduction to Coding Quantum Algorithms: ATutorial Series Using Qiskit Daniel Koch∗, Laura Wessing, Paul M. Alsing 2019 pp. 5-29.

29. https://ieeexplore.ieee.org/document/9274649

30. https://ieeexplore.ieee.org/document/9277798

31. T. Draper (2000), Addition on a quantum computer, quant-ph/0008033.

32. C. Moore and M. Nilsson (2012), Parallel quantum computation and quantum codes, SIAM J. Comp., 31, pp. 799-815.

33. L.M.K. Vandersypen, M. Steffen, G. Breyta, C.S. Yannoni, M.H. Sherwood, and I.L. Chuang (2018), Experimental realization of Shor’s quantum factoring algorithm using magnetic resonance, Nature, 414, pp. 883-887.

Додаток 1

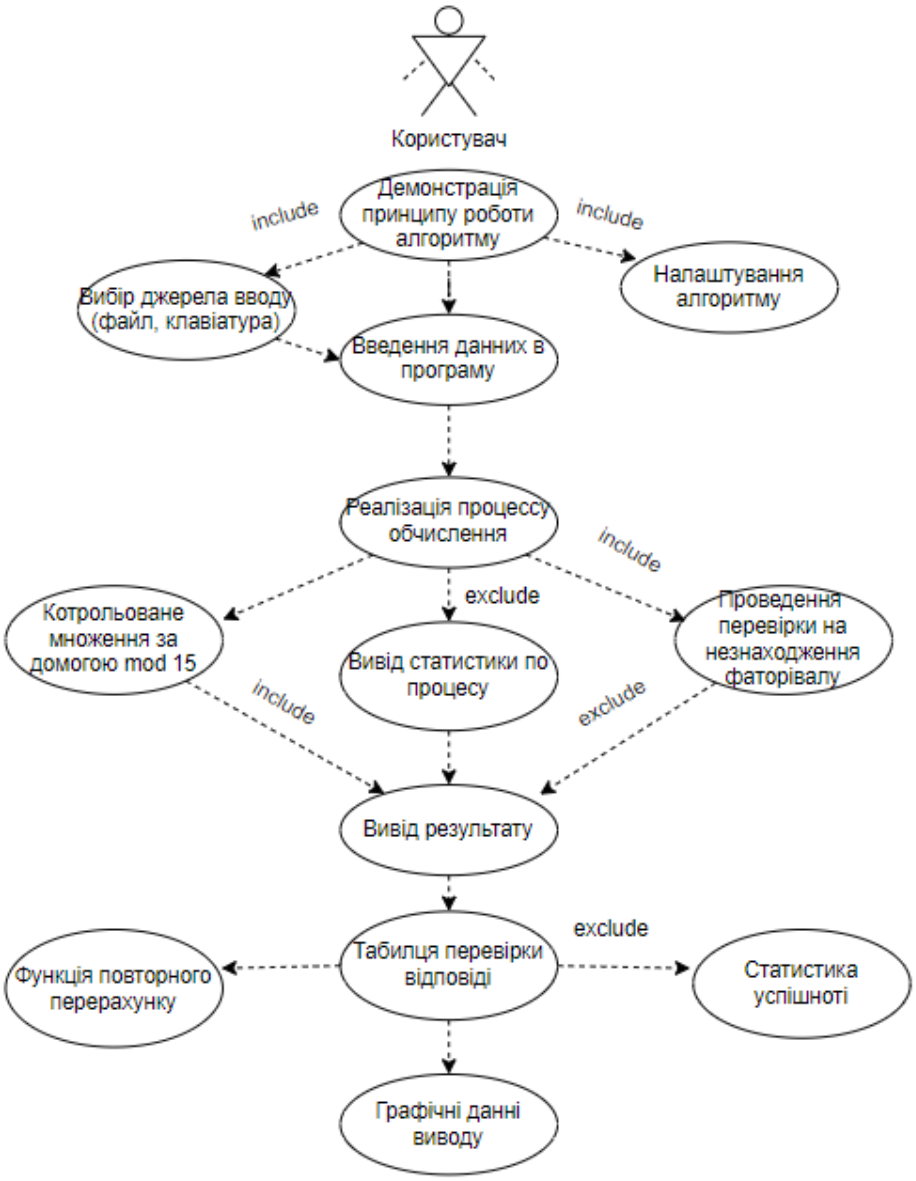
Система моделювання квантових обчислень

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2022 р.



					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Баланюк С.О.			**Система моделювання квантових обчислень** Структурна схема системи	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Луцький Г.М.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-з81	
Н. Контр.		Сімоненко В. П.					
Затвердив							

Додаток 2

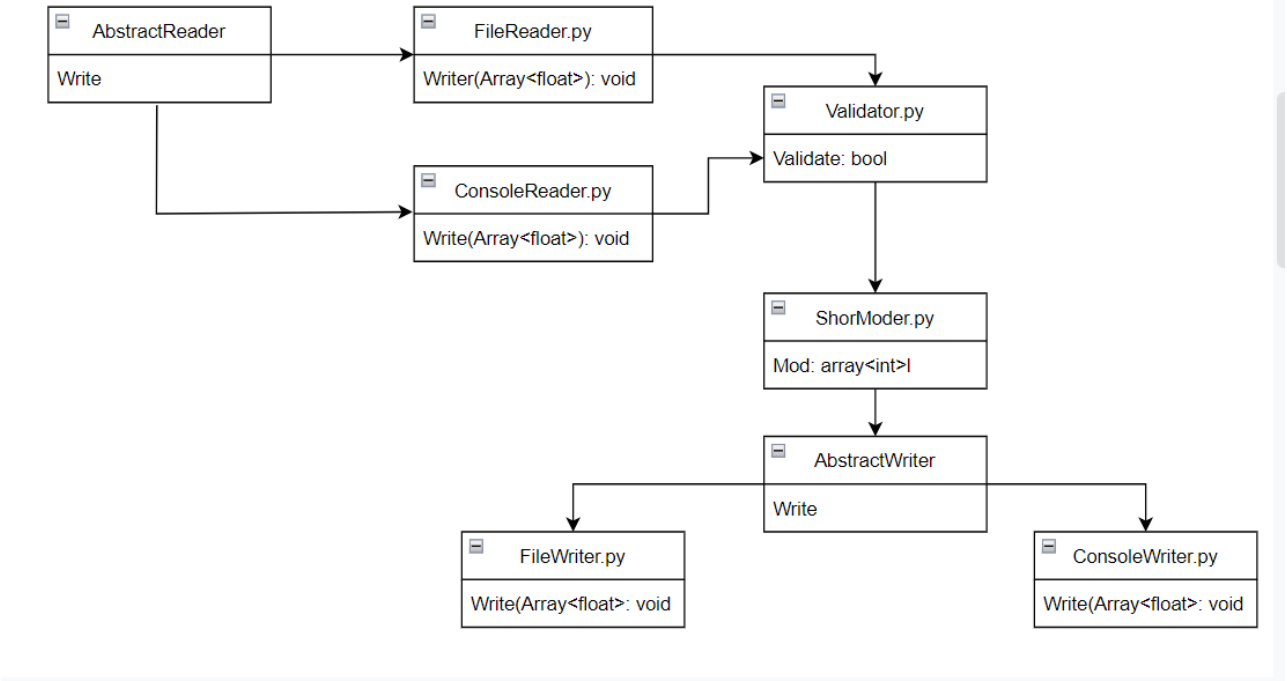
Система моделювання квантових обчислень

Функціональна схема (діаграма класів)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2022 р.



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Баланюк С.О			**Система моделювання квантових обчислень** Функціональна схема (діаграма класів)	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Луцький Г.М.				
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-з81
Н. Контр.		Сімоненко В. П.				
Затвердив						

Додаток 3

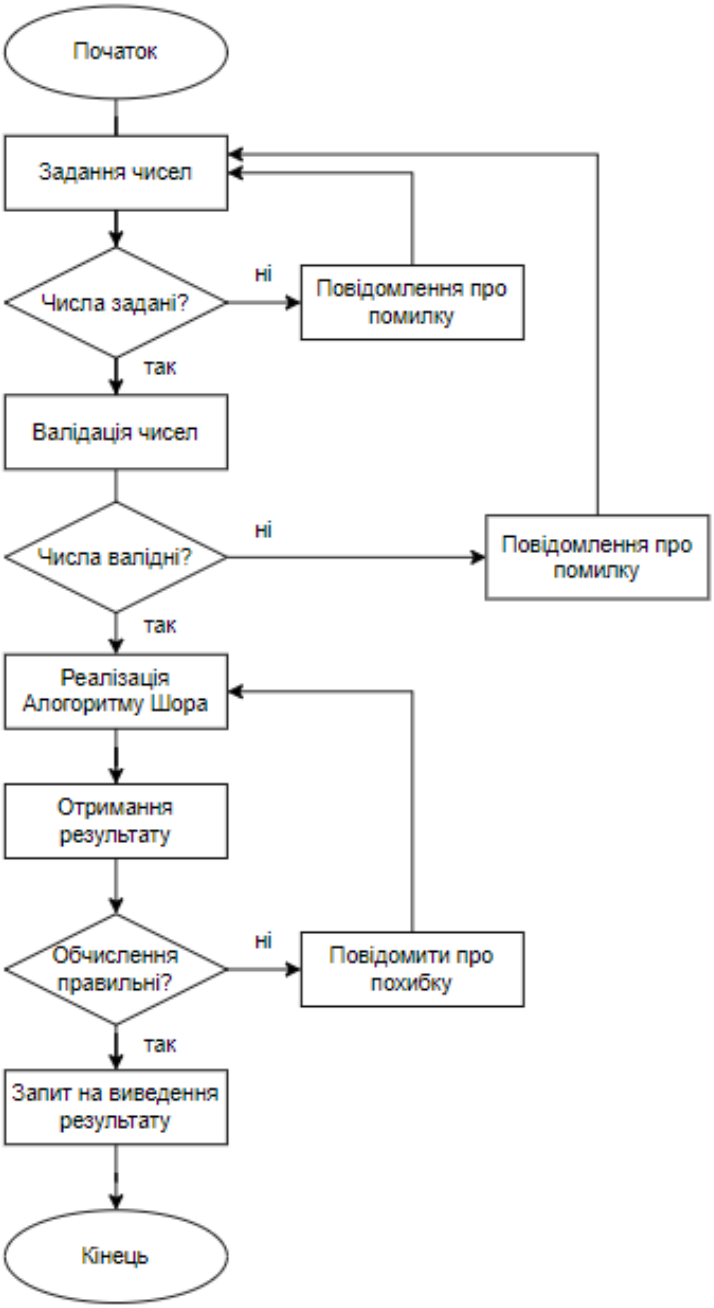
Система моделювання квантових обчислень

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2022 р.



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Баланюк С.О			**Система моделювання квантових обчислень** Алгоритм дій програмного забезпечення	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Луцький Г.М				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-з81
Н. Контр.		Сімоненко В. П.				
Затвердив						